VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Simulátor řízení aut s využitím platformy V-REP

Car Driving Simulator Based on Platform V-REP

2019 Bc. Krzysztof Heczko

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Krzysztof Heczko**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Simulátor řízení aut s využitím platformy V-REP
Car Driving Simulator Based on Platform V-REP

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit simulátor řízení auta po definované trati s využitím platformy V-REP s možností napojení umělé inteligence z různých programovacích jazyků. Účelem práce je především vylepšení simulátoru využívaného v předmětu Neuronové sítě. Součástí práce jsou také experimenty s různými umělými inteligencemi.

Simulátor umožní:
1. Získávat informace o okolí auta (překážky, vzdálenost od tratě).
2. Rozlišování statických a pohybujících se překážek.
3. Možnost předzpracování získaných dat a následné zpracování odesílaných dat zpět do simulátoru.

Práce bude obsahovat:
1. Popis možností simulátoru V-REP.
2. Implementaci výše popsané funkcionality.
3. Experimenty a vyhodnocení výsledků.
4. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.

Seznam doporučené odborné literatury:

[1] ROJAS, Raúl a FOREWORD BY JEROME FELDMAN. Neural networks: a systematic introduction. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. ISBN 9783642610684.
[2] V-rep: virtual robot experimentation platform [online]. [cit. 2016-04-04]. Dostupné z: http://www.v-rep.eu/
[3] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025

Dále dle pokynů vedoucího práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 26. dubna 2019 .....

Rád bych poděkoval svému vedoucímu diplomové práce, panu Ing. Davidu Ježkovi, Ph.D., za vedení, hodnotné rady a pomoc při její tvorbě.

Rovněž děkuji své rodině, přátelům a známým za jejich morální podporu.

Abstrakt

Obsahem této diplomové práce je popis implementace programu, který umožňuje řízení modelu automobilu ovládaného neuronovou sítí, simulované pomocí platformy V-REP. Úvodní část práce je věnovaná neuronovým sítím. Následuje představení a detailní popis možností programu V-REP. Další část se zaobírá samotnou aplikací, která zajišťuje komunikaci, získávání a zpracování dat ze simulátoru včetně řízení vymodelovaného automobilu pomocí umělé inteligence. Závěrečná část se zaměřuje na možnosti rozšíření použitých neuronových sítí.

Klíčová slova: V-REP, neuronová síť, umělá inteligence, simulace, model automobilu

Abstract

The content of this thesis is a description of the implementation of a program that controls a car model by a neural network, simulated using the V-REP platform. The first part of the thesis is devoted to neural networks. It is followed by an introduction and a detailed description of the V-REP program. The next part is concerned with the application ensuring communication itself, and the acquisition and processing of the data from the simulator, including the controlling of the car using artificial intelligence. The last part focuses on extensions of the used neural networks.

Key Words: V-REP, neural network, artificial intelligence, simulation, car model

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
B∅	–	BlueZero
IP	–	Internet Protocol
ODE	–	Open Dynamics Engine
ROS	–	Robot Operating System
UML	–	Unified Modeling Language
V-REP	–	Virtual Robot Experimentation Platform
XML	–	Extensible Markup Language

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Cílem diplomové práce je vytvoření simulátoru řízení automobilů po předem vyznačené trase, jehož řidičem (řídící jednotkou či mozkem) je neuronová síť. K simulaci jízdy samotné je použita platforma pro modelování, ovládání a simulaci robotů V-REP.

Popisovaný scénář je již aktuálně řešen simulátorem *Neural Racer*, který byl dříve vyvinut v rámci jiných závěrečných prací a je plně přizpůsoben požadavkům předmětu *Neuronové sítě*. Ve zmiňovaném předmětu studenti navrhují, implementují a posléze požívají neuronovou síť k řízení automobilu. Hotové sítě jsou pak užity k souboji vozidel na trati, kde lze pomocí simulace jízdy v reálném čase zjistit, jak si která z nich vede, zda byly přijatelně navržené trénovací množiny jednotlivých sítí a zda jsou schopné dostatečně přesně reagovat na data, jako například vzdálenost od tratě či překážky, zasílané ze simulátoru.

Obsahově je práce rozdělená do několika hlavních segmentů. První část pojednává o pojmu simulace obecně, po které následuje stručný teoretický popis neuronových sítí. Nejobsáhlejší pasáž je věnována simulátoru V-REP a detailnímu seznámení s jeho možnostmi. Další části se už zabírají samotným programem k řízení automobilů pomocí umělé inteligence. Nejprve je vylíčena příprava na straně simulátoru včetně modelovaní tratě a vozu, po které přichází na řadu samotný popis aplikace použité k řízení programu V-REP a simulace jako takové. V poslední části jsou analyzovány možnosti dalšího rozšíření použitých neuronových sítí.

Hlavním cílem této diplomové práce je tedy určité vylepšení simulátoru *Neural Racer* pomocí platformy V-REP, která je pro simulace přímo předurčena.

2 Simulace

2.1 Pojem simulace

Simulací rozumíme napodobování věcí, stavů či procesů reálného světa. Je využívána v nepřeberném množství odvětví, které zahrnují například modelování přírodních, lidských ale i za hranice naší planety sahajících systémů. Jejím cílem je získávat poznatky, data či fakta. Dále se můžeme bavit kupříkladu o simulacích technologických pro optimalizaci výkonu, správné testování anebo vzdělávání, simulacích ve zdravotnictví anebo sociálních oborech. Odvětví, kde všude se simulací aktuálně využívá, je enormní množství a dalo by se prakticky říct, že cokoliv, co funguje (nefunguje) reálně, lze převést do simulované podoby, kde se ono zmiňované cokoliv bude v ideálním případě chovat naprosto stejně.

2.2 Historie

Z historického hlediska se simulace využívané v různých sektorech rozvíjely nezávisle na sobě až do 20. století. Později se na celou problematiku začalo hledět více systematicky, k čemuž určitě přispěl i prudký vývoj počítacích strojů a jejich využití prakticky ve všech oblastech. Tyto skutečnosti vedly ke sjednocení názorů v dané oblasti a systémovému pohledu na problematiku simulace obecně.

2.3 Využití

Odpovědí na otázky, proč jsou simulace v dnešní době tak četně využívané a z jakého důvodu se vyplatí věci napodobovat, je mnoho. Překrývají obrovské spektrum výhod od finanční náročnosti reálných testů až po bezpečnost při jejich realizaci.

Zmiňované dva faktory si můžeme nastínit na jednom příkladu, a to z oblasti vývoje automobilů. Všechny nové modely aut, které automobilky vyvíjejí a následně posílají do provozu, musejí nejprve přejít nesčetným množstvím testů. Jedním z nich je i takzvaná nárazová zkouška neboli *crash test*. Právě u tohoto typu zkoušek jsou aktuálně v automobilkách využívané simulace, které jsou na tak vysoké úrovní, že prakticky kopírují opravdový náraz automobilu do zdi. Umí věrně napodobit chování skeletu automobilu při nárazu, kroucení plechů a deformaci karosérie, pevnosti součástek či vazeb, ba dokonce lze pozorovat pohyb pasažéru uvnitř vozidla, a imitovat tak figuríny které jsou k tomuto účelu použity v testu reálném. Nutno však podotknout, že simulace nárazového testu pouze předchází jeho opravdovou variantu, která je k uvedení automobilu do provozu v závislosti na trhu nutná. Nicméně už při simulaci jsou strojní inženýři schopni odhalit neduhy, na které by se jinak přišlo až při zničení vozidla.

Nepočítáme-li s tím, že k věrnému napodobení objektu, procesu anebo například stavu, potřebujeme dostatečně znát vzor ze kterého vycházet, pak pomocí simulací můžeme jednoduše zrcadlit chování objektů reálného světa bez toho, aniž bychom k tomu tyto objekty vůbec potřebovali. V tom okamžiku se otevírá spousta možností, kam můžeme směřovat. Lze nejen věrně

napodobovat a zkoušet různé možnosti simulovaných předmětů, ale rovněž zacházet za hranu jejich fyzikálních vlastností a přirozených procesů. „Výš, dál a rychleji“ je slovním spojením, které přesně vystihuje možnosti simulace. Lidské fantazii se pak již meze nekladou. Různým nastavením simulačních parametrů anebo vlastností objektů lze pozorovat, jak moc jinak by se věci chovaly, kdyby stoprocentně neodpovídaly realitě. I při takových pokusech se určitě jednou za čas narazí na chování, které kdysi mohl člověk zavrhnout už jenom z principu, který nedával smysl, načež při simulaci se ukáže, že výsledkem je diametrální opak.

3 Neuronové sítě

3.1 Specifikace

Neuronové sítě jsou v informačních technologiích inspirované reálnými, biologickými neuronovými sítěmi. Předpokládá se, že umělé sítě by si ze stránky základních principů či chování měly počínat stejně anebo minimálně podobně jako jejích biologické vzory. Snažit se věrně vymodelovat umělý mozek, který by přesně kopíroval ten lidský, je ovšem věc nereálná, respektive přinejmenším těžce dosažitelná, kde se problémem stává třeba už jenom počet neuronů anebo způsob, jakým jsou vzájemně propojeny [1].

Při provádění výpočtů umělé neuronové sítě je využíváno takzvané distribuované paralelní zpracování dat. Předávání informací, jejích úprava či ukládání probíhá pomocí sítě v celé její velikosti.

Znalosti neuronových sítí jsou ukládány pomocí síly vazeb mezi jednotlivými neurony. Vazby mezi neurony, které vedou ke správné odpovědi se posilují, kdežto u špatně vyhodnocených výsledku jsou naopak oslabovány.

Základním rysem neuronových sítí, kterým se podstatně liší od algoritmizace, je jejích trénování. U modelování algoritmů se soustředíme na správné převedení vstupních hodnot na hodnoty výstupní. Při tvorbě umělých neuronových sítí si můžeme dovolit tento důležitý krok vynechat, jelikož jej samy sítě ve svém principu nepotřebují. To, jakým způsobem se bude vstupní množina dat transformovat na množinu dat výstupních, určuje fáze trénování, jinak taky učení sítí, které je založeno na poskytnutí předem vytvořené sady vzorků, řekněme správných příkladů či vyhodnocení, které popisují řešenou problematiku.

3.2 Neuron

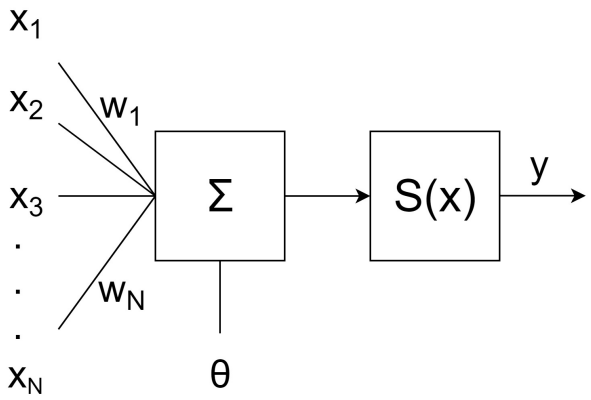
Základními výpočetními jednotkami sítě jsou neurony, které jsou dány jejich biologickými vzory a jejichž kombinací jsme schopni navrhnout a postavit komplexnější strukturu, tedy neuronovou síť. Modelů neuronů je popsaná celá řada a mezi ty nejznámější se řadí vzor popsaný McCullochem a Pittsem:

$$y = S(\sum_{i=1}^{N}(w_i x_i) + \theta)$$

kde:

- x_i jsou vstupy neuronu
- w_i jsou synaptické váhy
- θ je práh
- $S(x)$ je přenosová funkce neuronu
- y je výstup neuronu.

Velikostí vah w_i je vyjadřována uložená zkušenost neuronu. Se zvyšující hodnotou váhy nám roste i důležitost odpovídajícího vstupu. Prahem θ rozumíme minimální velikost vnitřního potenciálu nutnou k aktivaci daného neuronu. Za předpokladu, že práh překročen není, setrvává neuron v pasivním, tedy utlumeném stavu. Grafická prezentace modelu neuronu je znázorněná na obrázku 1.



Obrázek 1: Model umělého neuronu

3.2.1 Perceptron

Jedná se o jeden z nejdůležitějších modelů neuronu dnes používaných. Jeho vnitřní potenciál je definován jako vážený součet vstupujících signálů. Pokud je tímto potenciálem překonána prahová hodnota neuronu, dochází k jeho excitaci na hodnotu 1. Pokud prahová hodnota zmiňovaným součtem překonána není, neuron setrvává v utlumeném stavu, reprezentovaným 0. Pro zjednodušení je na nultý vstup zavedený stály neuron se stavem excitace $x_0 = 1$ a váhou $w_0 = -\theta$.

3.2.2 Spojitý perceptron

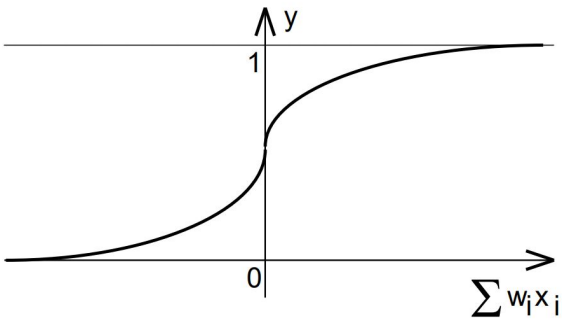
Hodnota excitace neuronu je dána přenosovou funkcí, rovněž označovanou jako aktivační nebo excitační. V případě spojitého perceptronu se jedná o funkci ve tvaru sigmoidy. Zmiňovanou funkci, jejíž průběh je znázorněn na obrázku 2, lze vyjádřit matematickým vztahem:

$$y = S(z) = \frac{1}{1 + e^{-\lambda z}}$$

kde:

- $S(z)$ je aktivační funkce neuronu
- z je vnitřní potenciál neuronu

- λ je strmost sigmoidu.



Obrázek 2: Přenosová funkce neuronu [1]

Samotný vnitřní potenciál neuronu je pak vyjádřen jako:

$$z = \sum_{i=0}^{N} (w_i x_i)$$

Z uvedených poznatků vyplývá, že excitace neuronu se pohybuje v intervalu 0 až 1, kde hodnota 1 označuje jeho plnou excitaci a na druhé straně hodnota 0 stav jeho utlumení. Za předpokladu že se vnitřní potenciál neuronu blíží hodnotě $+\infty$, dochází k jeho plné excitaci, naopak v případě, že se vnitřní potenciál blíží $-\infty$, neuron je úplně utlumen.

3.2.3 Adaptace perceptronu

Pokud nastavujeme váhy perceptronu tak, aby byl schopen rozpoznávat a správně reagovat na jednotlivé vstupy, bavíme se o procesu jeho učení. Cílem takového postupu je co možná nejpřesnější nastavení vah neuronů, které jde ruku v ruce s věrnými výsledky, tedy korektními odpověďmi perceptronu na předložené informace. Tato fáze učení je rovněž nazývána jako adaptivní, jelikož se neuron (respektive neuronová síť) přizpůsobuje vstupům, které mu předkládáme. Jedním z nejznámějších algoritmů adaptování perceptronu je Hebbovo pravidlo (Donald Olding Hebb byl kanadský psycholog, který působil v oboru neuropsychologie) a skládá se z této posloupnosti kroků:

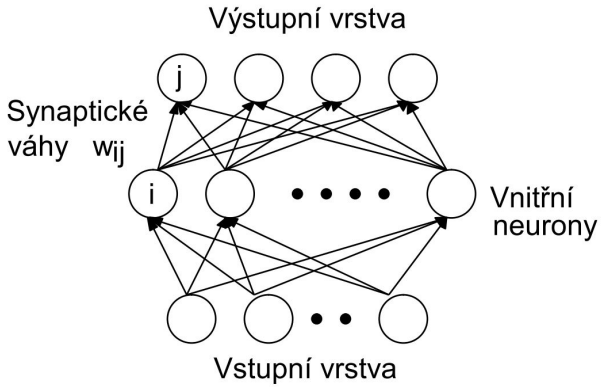
1. Inicializace vah a prahu perceptronu pomocí náhodných malých čísel.
2. Předložení vstupů a očekávaného (požadovaného) výstupu.
3. Určení reálné odezvy.
4. Samotná adaptace vah v závislosti na tom, zda byl skutečný výsledek správný či naopak. Tento krok algoritmu lze ještě dále modifikovat například určením razance změn hodnot adaptovaných vah.

3.3 Vícevrstvá neuronová síť

3.3.1 Struktura

Celek propojených neuronů tvoří samotnou neuronovou síť. Jedním z nejrozšířenějších a pravděpodobně i nejznámějších typů sítí spojitých perceptronů jsou vícevrstvé sítě. Takovou neuronovou síť tvoří minimálně tři vrstvy neuronů: vstupní, výstupní a přinejmenším jedna vnitřní.

Jednotlivé sousední vrstvy spojitých perceptronů jsou pak mezi sebou spojené takzvaným úplným propojením neuronů, což znamená, že každý neuron nižší vrstvy je spojen se všemi neurony sousední vrstvy vyšší úrovně. Vícevrstvá síť perceptronů včetně spojení neuronů mezi jednotlivými vrstvami je znázorněná na obrázku 3.



Obrázek 3: Vícevrstvá neuronová síť [1]

3.3.2 Feedforward

Zpracování informace vícevrstvou neuronovou sítí můžeme rozdělit na dva typy: dopředné a zpětné. Dopředné šíření signálů neboli *feedforward* probíhá takto:

1. Na začátku jsou excitovány neurony vstupní vrstvy, a to hodnotami v rozmezí 0 až 1.
2. Excitace z prvního kroku jsou pomocí vazeb přivedené k sousední, vyšší vrstvě a následně jsou zesíleny či naopak zeslabeny na základě synaptických vah.
3. Každý z neuronů vyšší vrstvy provede sumaci těchto upravených signálů z vrstvy nižší a posléze je excitován na úroveň, která je dána jeho aktivační funkcí.
4. Takto signály putují skrz všechny vnitřní vrstvy neuronové sítě až k vrstvě výstupní, kde pak získáme excitační stavy všech neuronů, které uvedenou výstupní vrstvu tvoří.

Uvedeným způsobem lze získat odezvu neuronové sítě na příchozí impuls daný excitací vstupní vrstvy, a jak lze vytušit, základem správného rozhodnutí sítě jsou korektně nastavené synaptické váhy, kde proces jejích seřizování opět nazýváme adaptací, tedy učením.

3.3.3 Trénovací množina

K tomu, abychom neuronovou síť naučili správně reagovat na vstupní signál, potřebujeme takzvanou trénovací množinu, která obsahuje prvky popisující danou problematiku, a neméně pak metodu, kterou jsme schopní takové elementy do sítě upevnit v podobě nastavených hodnot synaptických vah.

Trénovací množina je sada dat, která se skládá ze vstupního vektoru informací a k němu odpovídajícímu vektoru výstupnímu. Abychom si byli jistí správností adaptace vícevrstvé neuronové sítě, obvykle potřebujeme, aby zmiňovaná sada dat byla značně veliká a přesná. Čím větší trénovací množina je, tím přesnější bude učení. Je zřejmé, že enormní velikosti trénovacích sad korespondují s časovou náročností samotné adaptace. V souvislosti s trénovací množinou zmiňme ještě jednu kolekci dat využívanou při učení neuronových sítí, a to sadu testovací. Jedná se o množinu, která by se neměla shodovat se sadou trénovací, ale rovněž obsahuje část korektních dat, pomocí kterých můžeme určit kvalitu adaptace. Pokud máme k dispozici pouze jednu soupravu informací určenou k učení, ale chceme si být jistí korektností adaptované sítě, nabízí se takovou množinu rozdělit v určitém poměru na dvě části, kde jednou z nich je síť trénována (tedy jsou nastavovány synaptické váhy neuronů), a na druhé si, například dopředným šířením, můžeme ověřit kvalitu výstupu. Trénovací sadu T můžeme definovat jako množinu uspořádaných dvojic:

$$T = \Big\{ \Big\{ I_1, O_1 \Big\} \quad \Big\{ I_2, O_2 \Big\} \quad \dots \quad \Big\{ I_p, O_p \Big\} \Big\},$$

$$I_i = \Big[i_1 \quad i_2 \quad \dots \quad i_k \Big], \quad i_j \in \Big\langle 0, 1 \Big\rangle,$$

$$O_i = \Big[o_1 \quad o_2 \quad \dots \quad o_l \Big], \quad o_j \in \Big\langle 0, 1 \Big\rangle$$

kde:

- p je počet vzorů trénovací množiny
- I_i je vektor excitací vstupní vrstvy tvořené k neurony
- O_i je vektor excitací výstupní vrstvy tvořené l neurony
- i_j, o_j je excitace *j-tého* neuronu vstupní, respektive výstupní vrstvy.

3.3.4 Backpropagation

Jak již bylo uvedeno výše, jednou z metod zpracování dat pomocí vícevrstvé neuronové sítě je zpětné šíření. Zmiňovaná metoda se nazývá *backpropagation*, a umožňuje adaptaci neuronové sítě nad danou trénovací sadou. Na rozdíl od již popisovaného dopředného průchodu sítí se šíří od neuronů nejvyšší úrovně směrem k vrstvám nižším. V jednotlivých krocích vypadá takto:

1. Vektorem I_i *í-tého* prvku trénovací množiny excitujeme neurony vstupní vrstvy na odpovídající úroveň.

2. Pomocí dopředného šíření dovedeme signál až k výstupní vrstvě.

3. Požadovaný výsledek daný vektorem O_i *í-tého* prvku adaptační sady porovnáme se skutečnou odezvou neuronové sítě.

4. Rozdíl mezi těmito výsledky nám definuje chybu neuronové sítě. Tu vracíme zpět do sítě formou úpravy synaptických vah mezi vrstvami neuronů na jednotlivých úrovních směrem od nejvyšších k nejnižším tak, aby při následující odezvě byla chyba menší.

5. Po vyčerpání celé trénovací množiny je vyhodnocená celková chyba pomocí všech vzorů trénovací sady. Pokud je zmíněná chybovost vyšší než požadovaná, celý proces se opakuje.

3.3.5 Generalizace

Síť, která by nám uměla správně vyhodnocovat pouze ta data, které byly použity k její adaptaci, by nám zřejmě příliš užitku nepřinesla. Při aplikování neuronových sítí na danou problematiku nám totiž jde hlavně o to, aby uměly korektně reagovat na data či impulsy, která jsou podobná těm trénovacím, ale totožnost v nich budeme hledat jen obtížně. Bavíme se o schopnosti generalizace čili zobecnění, jinak řečeno vlastnosti sítě odpovídat na jevy, které nebyly součástí jejího učení, ale dají se z něj však určitým způsobem odvodit. Proto by se při učení mělo dávat pozor na takzvané přetrénování sítě, což je stav, kdy má vícevrstvá neuronová síť nastavené váhy vůči trénovací sadě tak dokonale, že umí bezchybně rozpoznat adaptivní množinu, nicméně data podobná těm trénovacím, které bychom pomocí ní chtěli analyzovat, jí jsou cizí. Tento jev je typický při zbytečně dlouhém a precizním učení. Naopak, u příliš odbyté fáze adaptace vah, nemá dotyčná neuronová síť možnost naučit se ani trénovací množinu, načež po ní chceme vracet smysluplné výsledky pro obecná vstupní data. Z toho vyplývá, že celý proces učení je jakýmsi kompromisem mezi těmito dvěma radikálními stavy a že je zapotřebí síť učit tak, aby bylo dosaženo dostatečného natrénování adaptační sady nerozlučně se schopností generalizace neznámých dat.

4 V-REP

4.1 Představení simulátoru

V-REP je open source robotický simulátor vyvíjený švýcarskou společností Coppelia Robotics. Jedná se o program s integrovaným vývojovým prostředím založený na architektuře distribuovaného řízení, kde každý objekt nebo model může být řízen separátně, a to pomocí *embedded* scriptů, pluginů či například vzdálených API klientů. Toto činí z platformy V-REP velice univerzální nástroj, který je ideální pro simulaci vícera robotů v rámci jedné scény, kde můžeme jednoduše analyzovat a upravovat jejich vzájemnou interakci. Ovladače modelů, tedy *controllery* jednotlivých robotů, můžeme zapsat pomocí programovacích či skriptovacích jazyků jako jsou C, C++, Python, Java, Lua, Matlab anebo Octave [2].

V-REP je používán například k rychlému a snadnému vývoji algoritmů, simulacím automatizace, prototypování a verifikaci anebo vzdálenému monitorování. Jedná se multiplatformní software, který umožňuje vytváření přenosného, škálovatelného a snadně udržovatelného obsahu. Simulátor a simulace jsou plně přizpůsobitelné a dostupné pomocí šesti programovacích přístupů, které jsou vzájemně kompatibilní. Obsahuje čtyři fyzikální enginy: Bullet Physics, ODE, Newton a Vortex Dynamics, pro dynamické kalkulace, simulaci fyziky a interakci mezi objekty. Mezi další vlastnosti simulátoru patří například detekce kolizí, počítání minimální vzdálenosti mezi objekty, dynamická simulace částic (kupříkladu napodobení proudu vzduchu či vody), senzorů přiblížení anebo snímačů.

4.2 Uživatelské rozhraní

Aplikace V-REP je složena z více elementů. Mezi hlavní, viditelné na obrázku 4, patří:

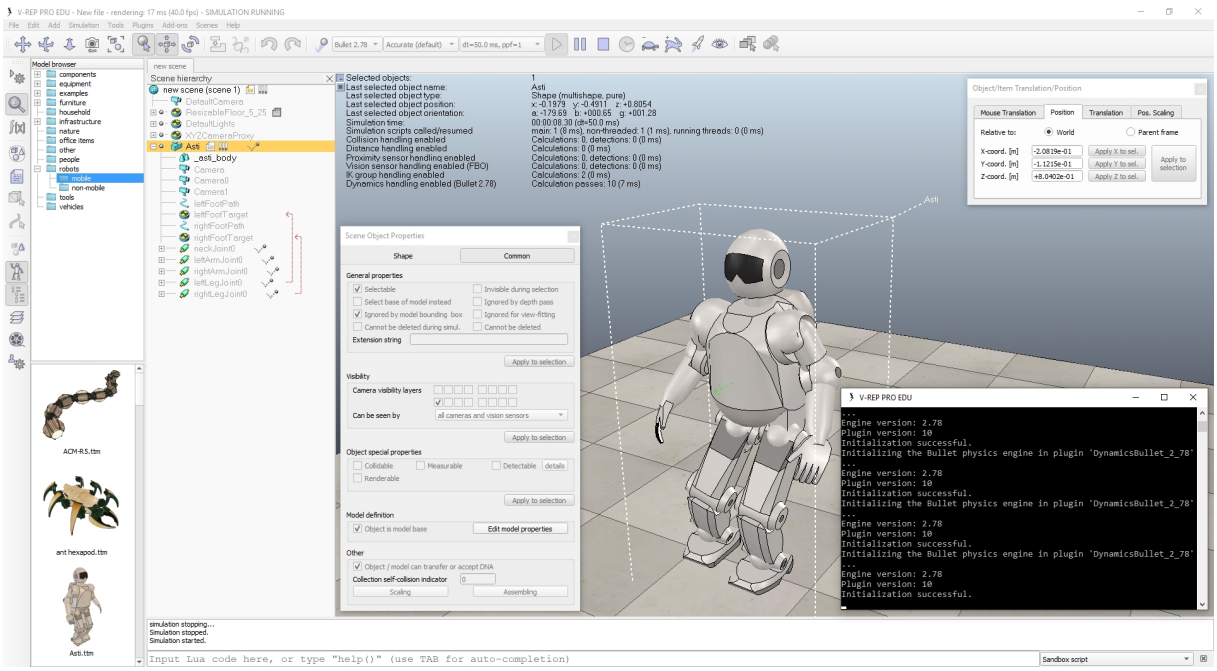
- Konzolové okno – vytváří se pokaždé když je program spuštěný pod operačním systémem Windows a ve výchozí konfiguraci je skryto, nicméně toto nastavení lze přizpůsobit. V operačním systému Linux musí být samotný V-REP spuštěný z konzole, která zůstává viditelná po celou dobu běhu aplikace. Konzolové okno zobrazuje, které pluginy jsou načítány a zda byla jejich inicializace úspěšná či nikoliv. Není interaktivní a slouží pouze jako výstup, do kterého může přistupovat (vypisovat) i uživatel.

- Aplikační okno – je hlavním oknem aplikace. Je použito k zobrazení, editaci, simulaci a interakci se scénou.

- Dialogy – mimo aplikační okno může uživatel editovat vlastnosti a seřizovat scénu, respektive její objekty, přizpůsobováním dialogových nastavení anebo parametrů. Každý dialog seskupuje soubor souvisejících funkcí, anebo funkce, které lze aplikovat na objekty stejného typu. Obsah dialogu může být kontextově závislý, například na stavu vybraného objektu.

Při spuštění programu, aplikace inicializuje jednu výchozí scénu. Jedná se o prázdnou rovinu bez jakýchkoliv objektů. Uživatel může mít otevřeno více scén paralelně, kde každá z nich sdílí

aplikační okno a dialogy se scénami jinými, nicméně zobrazený obsah včetně dialogů bude vždy odpovídat pouze té scéně, která je aktuálně aktivní. Jinými slovy, v rámci jednoho spuštění aplikace V-REP můžeme v daný čas pracovat pouze s jednou scénou, ačkoliv jich program dovoluje mít otevřených víc současně.



Obrázek 4: Uživatelské rozhraní robotického simulátoru V-REP

Výše popisované aplikační okno se dělí na další elementy. Mezi nejdůležitější z nich, všechny rovněž viditelné na obrázku 4, patří:

- Panel nástrojů – jehož odkazy (tlačítka) reprezentují nejčastěji používané funkce při práci s programem. Většinu z nich lze vyvolat i jiným způsobem, například volbou v menu. V základním nastavení programu se jedná o dva panely, svislý a vodorovný.

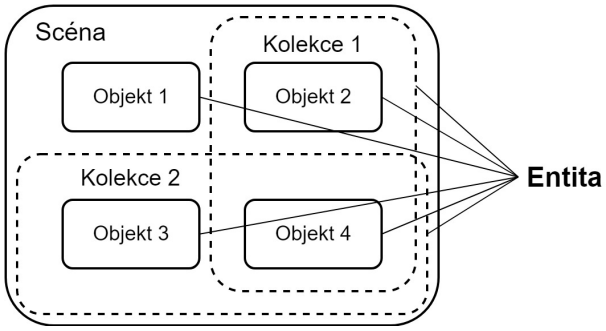
- Prohlížeč modelů – pomocí kterého můžeme procházet dříve vytvořené modely. Ve vrchní polovině se nachází složková struktura, spodní pak poskytuje vizualizaci modelů v rámci vybrané složky. Přetáhnutím náhledu do prostoru scény V-REP načte odpovídající model. Ve výchozím nastavení je prohlížeč viditelný.

- Hierarchie scény – jeden z nejdůležitějších elementů aplikačního okna při tvorbě a správě modelů načtených v rámci scény. Jedná se o nativně viditelný panel, který obsahuje všechen obsah scény, tedy všechny objekty, které jí tvoří. Jelikož jsou objekty scény ukládány v hierarchické struktuře, panel zobrazuje strom této struktury. Jednotlivé elementy můžou být rozvinuty či naopak schovány (zabaleny). Dvojím kliknutím na ikonu objektu se přistupuje k jemu odpovídajícímu dialogu, stejná akce směrovaná na jeho název zase umož-

ňuje přejmenování. Objekty můžou být pomocí hierarchie scény v její rámci přesouvány do jiných objektů, čímž se tvoří, respektive ruší, vztah předek-potomek.

4.3 Entita

Entitou v rámci aplikace V-REP rozumíme objekt scény anebo kolekci takových objektů. Vztah mezi nimi je zobrazen na obrázku 5.



Obrázek 5: Entita

4.3.1 Objekt

Hlavními elementy v aplikaci, které tvoří scénu jsou takzvané objekty scény. Jsou viditelné jak v pohledu scény samotné, tak v její hierarchii, kde je reprezentuje náhled specifický pro daný typ. Jednotlivé typy objektů, včetně jejích náhledu v hierarchii a grafického příkladu jsou znázorněny na obrázku 6.

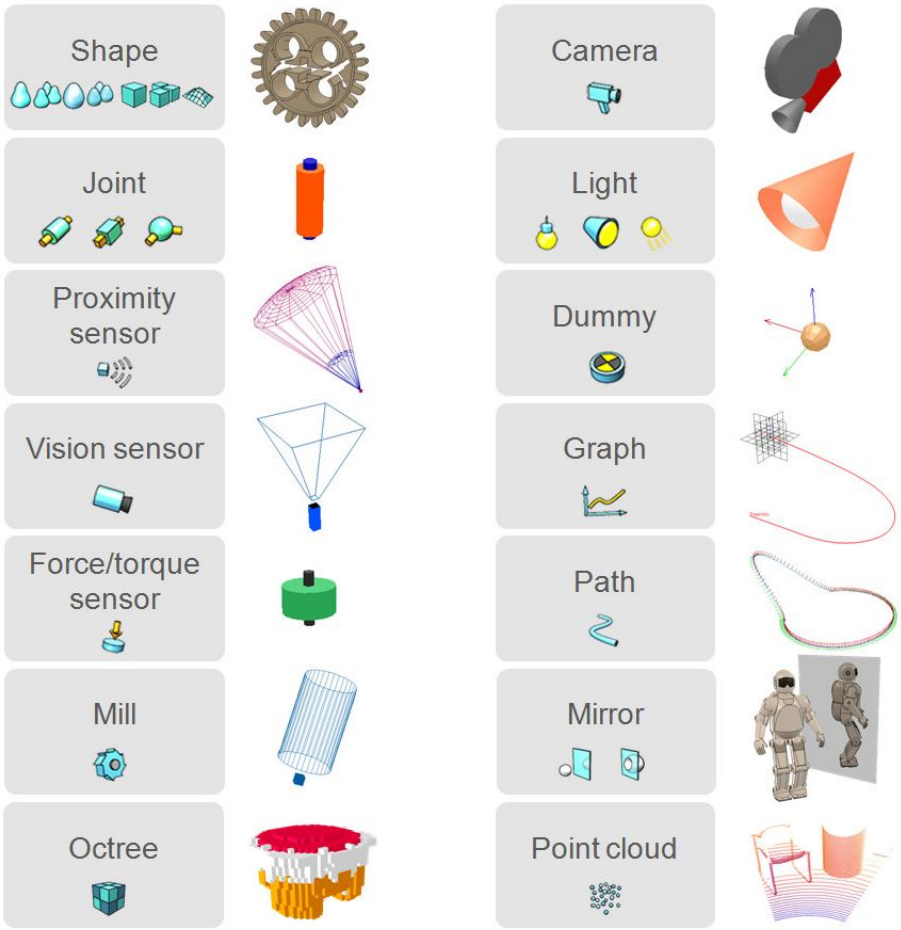
Některým typům objektů lze nastavit speciální vlastnosti, které umožňují jiným objektům anebo kalkulačním modulům jejich vzájemnou interakci. Bavíme se o:

- *Collidable* – detekce kolizí mezi jednotlivými objekty.
- *Measurable* – u měřitelných objektů lze počítat minimální vzdálenost mezi nimi.
- *Detecable* – objekt lze detekovat senzorem přiblížení.
- *Renderable* – objekt lze detekovat snímačem, tedy *vision* senzorem.
- *Viewable* – objekt může zobrazovat určitý obsah obrazu.

Každý objekt má pozici a orientaci v rámci scény. Tyto dvě vlastnosti patří mezi jeho konfigurační nastavení a lze je modifikovat.

Objekty lze spojovat v rámci stromové struktury scény, kdy mezi nimi vzniká již dříve vzpomínaný vztah předek-potomek. Pokud u takto vytvořené dvojice změníme pozici či orientaci předka, bude stejně reagovat i potomek v rámci jejich vzájemného nastavení. U komplikovanějších struktur složených z více objektů se tedy samo nabízí provazovat je právě tímto vztahem.

Změny pozice či orientace většího množství objektů pak lze modifikovat pouhým správným nastavením objektu umístěného výš ve stromové struktuře. V závislosti na možnosti vytváření vztahu mezi dvěma objekty, se u zmiňovaného nastavení orientace či umístění rozlišuje mezi jeho dvěma typy, a to nastavením globálním neboli absolutním a dále lokálním, které je relativní k pozici předka. Pokud z objektu B uděláme potomka objektu A, jeho globální nastavení pozice či orientace se nezmění, nicméně lokální už bude závislé na konfiguraci předka.



Obrázek 6: Typy objektů [2]

4.3.2 Kolekce

Kolekce je množina objektů scény definována uživatelem. Musí obsahovat přinejmenším jeden takový objekt a je považována za entitu (stejně jako pouhý objekt), což je rovněž viditelné z obrázku 5. Kolekce jsou užitečné při odkazování na skupinu objektů, které tvoří jeden logický celek. Program V-REP umožňuje provádění kalkulací a výpočtů nejen nad jedním objektem, ale i nad jejich skupinami, tedy kolekcemi. Jako příklad můžeme uvést kolekci A, která se skládá

z objektů *a*, *b* a *c* a objekt *B*. Při výpočtu detekce kolizí mezi kolekcí *A* a objektem *B* aplikace testuje, zdali kterýkoliv objekt z uvedené kolekce není v kolizi s objektem *B*.

Kolekce jsou *collidable*, *measurable*, *detecable* a *renderable*. Může tedy docházet k:

- Detekci kolizí mezi kolekcí a jinými entitami které mají tuto vlastnost.
- Měření minimální vzdálenosti mezi kolekcí a jinými entitami které jsou rovněž měřitelné.
- Detekci kolekce pomocí senzorů přiblížení.
- Detekci kolekce *vision* senzory, tedy snímači.

I když může kolekce nabývat zmíněné vlastnosti, nutně to neznamená, že všechny objekty, ze kterých se skládá, je musí mít rovněž. Například při určování minimální vzdálenosti jsou u jejího výpočtu zahrnuty pouze ty objekty kolekce, které tuto vlastnost mají aktivní. Obdobně to platí i pro další tři uvedené příznaky, kterými lze objekty v kolekci specifikovat. Pro usnadnění, aplikace V-REP umožňuje tyto vlastnosti v nastavení kolekce globálně přepsat pro všechny objekty v ní zahrnuté.

4.4 Scéna a model

Scény a modely jsou hlavními prvky simulace v aplikaci V-REP. Model je elementem scény a je tak i přímo označován. Scéna může obsahovat libovolné množství modelů. Jak scény, tak i modely můžou obsahovat prvky jako: objekty, kolekce, kolizní či vzdálenostní objekty, skupiny inverzní kinematiky, scripty a vlastní uživatelská rozhraní. Scény jsou v systému ukládány jako soubory s příponou **.ttt*, modely zas obsahují příponu **.ttm*. Oba typy souborů lze otevřít jejích přetáhnutím do okna aplikace, dvojím kliknutím anebo pomocí odpovídající volby v menu programu.

4.4.1 Scéna

Scény můžou obsahovat stejné elementy jako modely, navíc se ale skládají z prvků specifických přímo pro ně, a to:

- Prostředí – které definuje vlastnosti a parametry které jsou součástí scény, nicméně nepatří mezi její objekty. Tyto vlastnosti se nezahrnují při akci ukládání modelů, ale pouze při ukládání scény. Pomocí prostředí můžeme nastavit například barvu pozadí scény, parametry mlhy či nasvětlení.
- *Main* script – který řídí celou simulaci. Nativně má každá scéna jeden hlavní script, který obsahuje základní kód pro běh simulace, bez kterého by zmíněná simulace nedělala nic. Skládá se z funkcí, které jsou volány systémem a jakákoliv úprava hlavního scriptu není doporučená.

- Stránky a pohledy – které jsou odpovědné za zobrazení scény.

Při vytváření nové scény, je načtena scéna výchozí, která obsahuje kameru, nasvětlení, stránky a pohledy, prostředí, objekt roviny a hlavní script. Uživatel se může mezi scénami pohybovat pomocí hierarchie scény anebo pomocí odpovídajícího tlačítka panelu nástrojů.

4.4.2 Model

Jedná se o soubor objektů scény, které jsou postaveny (uloženy) v jedné stromové struktuře, kde první objekt v pořadí musí nést specifický příznak, kterým se dává najevo, že se jedná o základ modelu. V opačném případě, aplikace takovou skupinu objektů nepovolí coby model ani uložit, a jedinou možností k jejich zachování zůstane uložení scény. Pokud je dodržen správný postup vytváření modelu, u jeho ikony v hierarchii scény se zobrazí ikona míče (tag), která znázorňuje, že daný objekt je modelem a pomocí kterého lze přistupovat k dialogu jeho nastavení.

Jak již bylo řečeno výše, modely jsou jednotlivými elementy scény. K úspěšné simulaci modelu musí být tento soubor objektů umístěný právě v určité scéně. Pouze modelová simulace, tedy bez toho, aniž by byl daný model zmiňované scéně přiřazen, není možná. Jinak řečeno, model bez scény nemůže existovat (nepočítáme-li soubor uložený v systému).

4.5 Výpočetní moduly

V-REP disponuje výpočetní funkcionalitou, která není přímo uložená v objektu (jako například u senzorů přiblížení anebo snímačů), ale spíše takovou, kterou si uživatel může nastavit na určitý objekt, respektive objekty. Pomocí speciálních výpočetních modulů můžeme coby uživatelé získávat a sledovat doplňující informace o dění ve scéně či o samotné simulaci. Řeč je o:

- Modulu detekcí kolizí, který umožňuje sledování, zapisování a vizualizaci kolizí, které můžou nastat mezi dvěma *collidable* entitami. Jedná se o přesný interferenční výpočet, tedy vzájemné ovlivňování, prolínání nebo střetání jevů či hmoty. Modul pouze detekuje kolize, ale nereaguje na ně. Pomocí této detekce můžeme registrovat kolizní entity, které tvoří takzvaný kolizní pár. V průběhu simulace pak můžou být objekty které kolidují s jinými entitami například vizualizovány změnou jejich barvy či zanesením do grafu.

- Modulu výpočtu minimální vzdálenosti, pomocí kterého můžeme sledovat, zapisovat a vizualizovat nejkratší vzdálenost mezi dvěma měřitelnými entitami. Jedná se o přesný výpočet minimální vzdálenosti. Tento modul, stejně jako modul detekcí kolizí, umožňuje pouze měření bez přímé reakce na její velikost a výsledná vzdálenost v průběhů simulace může být rovněž zanesená do grafu či vizualizována pomocí přímky spojující dva objekty v bodech, které si jsou nejbližší.

- Modulu inverzní kinematiky, kterým lze řešit problémy inverzní anebo dopředné kinematiky.

- Dynamickém modulu, který, jak už název napovídá, umožňuje dynamicky simulovat objekty či modely k dosažení jejích vzájemné interakce (například reakce na kolize atd.).

Další podobné funkcionality lze dosáhnout importováním různých pluginů, kupříkladu můžeme uvést plugin pro plánování trasy.

Některé kalkulační moduly umožňují evidovat výpočetní objekty, které jsou definovány uživatelem. Ty se odlišují od objektů scény, nicméně jsou s nimi nepřímo spojeny, což znamená, že samy o sobě nemůžou existovat.

4.6 Dynamický modul

Dynamický modul aplikace V-REP v její nejnovější verzi disponuje podporou čtyř různých fyzikálních enginů. Uživatel se může svobodně rozhodnout, který z nich si přeje použít pro potřeby své simulace a jednoduše je zaměňovat. Důvodem této rozmanitosti podpory fyzikálních knihoven je, že simulace fyziky je komplexní úkol, který lze efektivně řešit různými stupni přesnosti, rychlosti nebo podporou rozličných funkcí. Jak již bylo řečeno v úvodu této kapitoly, jedná se o:

1. Bullet physics library [3] – open source fyzikální engine, mezi jehož vlastnosti patří 3D kolize anebo dynamika tuhých či měkkých těles (dynamika měkkých těles není aktuálně v simulátoru V-REP podporována). Je používaný ve hrách, často jako samotný herní fyzikální engine, nebo ve vizuálních efektech u filmů.

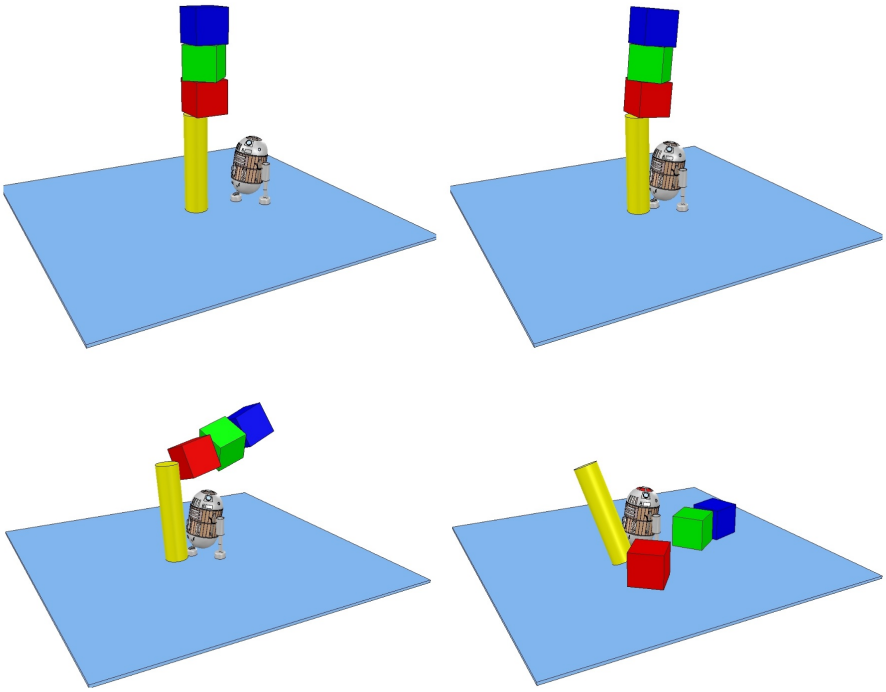
2. Open Dynamics Engine [4] – taky označován jako ODE, který se rovněž řadí mezi open source fyzikální enginy. Jeho stěžejními prvky jsou dynamika tuhých těles a kolize. Používá se v mnoha aplikacích a hrách, a stejně jako jeho ekvivalent uvedený výše často zastává roli herního fyzikálního enginu.

3. Vortex Dynamics – proprietární (closed source), komerční fyzikální engine poskytující vysoko-věrnostní fyzikální simulaci. Vortex disponuje reálnými parametry, které zrcadlí odpovídající fyzikální jednotky, a jsou použitelné pro veliký počet vlastností či funkcionalit, což z něj činí vysoce realistický engine s vysokou přesností simulované dynamiky. Používá se hlavně u průmyslových a výzkumných aplikací které vyžadují vysoký výkon. Vortex plugin který je použit v programu V-REP je založen na tzv. Vortex Studio Essentials, a k jeho používání je vyžadována registrace uživatele, která je nutná pro získání bezplatného licenčního klíče.

4. Newton Dynamics [5] – multiplatformní *life-like* fyzikální simulační knihovna využívaná ve hrách ale rovněž jako nástroj pro jakékoliv simulování fyziky v reálném čase. Plugin aktuálně využívaný v aplikaci V-REP je v beta verzi.

Všechny uvedené dynamické enginy disponují v programu V-REP nastavením, které je umožňuje konfigurovat. Takové nastavení je vždy specifické pro daný typ enginu.

Dynamický modul umožňuje simulování interakce mezi objekty na úrovní blížící se interakcím objektů reálného světa. Dovoluje objektům padat, kolidovat a odrážet se, ale rovněž pomocí něj můžeme s objekty manipulovat například pomocí robotických ramen, simulovat pohyb objektů na pásovém dopravníku či realisticky napodobovat jízdu modelů, například aut, terénem. Příklad simulace dynamiky je zobrazen na obrázku 7.



Obrázek 7: Simulace dynamiky

Na rozdíl od jiných, podobných simulačních softwarů, V-REP není čistě dynamickým simulátorem. Očima jeho vývojářů představuje spíš nějaký hybrid, který kombinuje kinematiku a dynamiku, aby nabídnul co nejlepší a nejpřesnější interpretaci pro různé simulační scénáře. V dnešní době se totiž fyzikální enginy stále spoléhají na mnoho aproximací a jsou relativně pomalé a nepřesné v porovnání se skutečností.

V aplikaci lze dynamicky simulovat jenom omezený počet objektů. Řadí se mezi ně tvary, spoje a silové senzory (*shapes*, *joints*, *force sensors*). Samozřejmě ale záleží na struktuře celé scény a vlastnostech objektů. Dynamicky simulované objekty jsou při simulaci v hierarchii scény označeny tagem (ikona znázorňující odrážející se míček), pomocí kterého je můžeme jednoduše odlišit od objektů statických hned na první pohled. Dvojím kliknutím na tento tag lze zobrazit informace související s dynamickým chováním objektu. Objekty, které by měly být dynamicky simulované, nicméně z nějakého důvodu to u nich učinit nelze, jsou rovněž označeny specifickou ikonou.

Tvary můžeme rozdělit do čtyř skupin v závislosti na jejich chování v průběhu dynamické simulace: *static*, *non-static*, *respondable* a *non-respondable*. Při takové simulaci nejsou statické

objekty ničím ovlivňovány, například tedy je relativní pozice statického potomka fixní vůči pozici předka, naopak na dynamické objekty budou působit fyzikální veličiny počínaje gravitací. Vlastnost *respondable* určuje, zda objekt reaguje, respektive nereaguje na jiné objekty. Toto nastavení objektů lze kombinovat. Uvažujeme-li, že v základní konfiguraci je objekt *non-respondable* a jeho předkem je prostředí, pak absolutní poloha statických objektů je permanentně stejná (pokud není její změna inicializována uživatelem, například programovou funkcí změny pozice) a nepůsobí na ně žádné fyzikální veličiny. Nastavením vlastnosti *respondable* na takovém objektu docílíme toho, že dynamické předměty se stejnou vlastností na ně budou reagovat (odrážet se, ležet na něm a tak dále). Dynamické objekty se stejným předpokladem se můžou v prostoru pohybovat. Pokud se přiblíží k jinému předmětu natolik, že dochází ke kolizi, pokračují dál skrz něj. Nastavením vlastnosti *respondable* na takovém objektu docílíme toho, že bude reagovat na jiné předměty se stejným znakem, například odražením od statických překážek anebo posunutím či shozením jiných dynamických objektů.

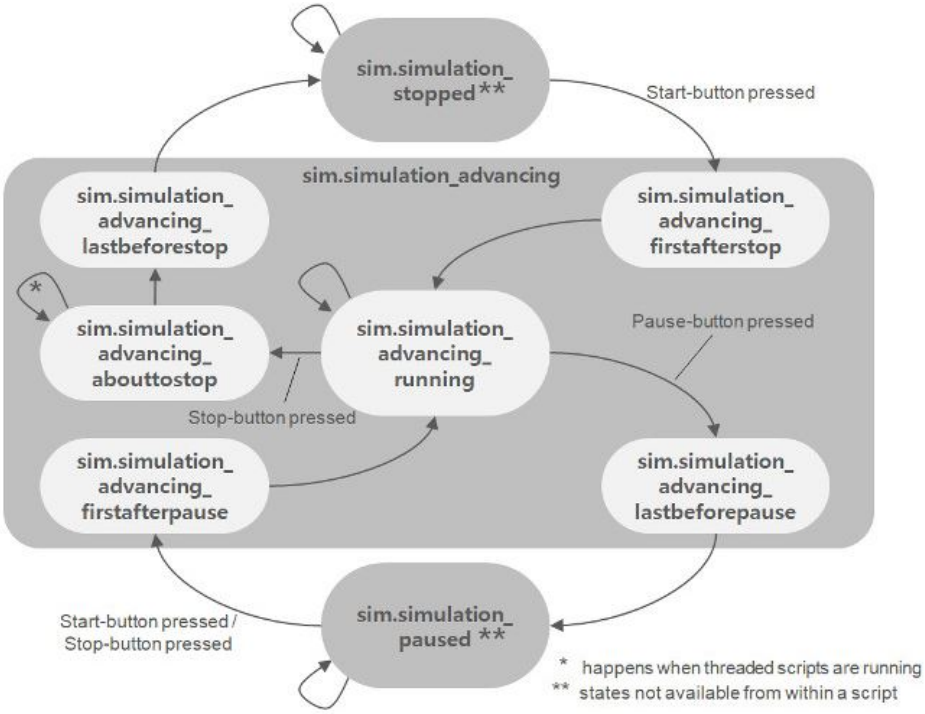
Dynamické tvary, které jsou umístěné nad povrchem roviny (tedy ve vzduchu), budou vlivem gravitace padat, a to za předpokladu, že nejsou proti tomuto chování nějakým způsobem vymezeny. Zmíněného vymezení docílíme pomocí takzvaných dynamických vazeb, které umožňují spojení dvou tvarů pomocí spoje anebo silového senzoru. U obou případů vyjmenovaných vazeb se vždy jedná o spojení, kde samotný spoj, respektive silový senzor, figuruje v roli prostředníka mezi tvarem (buď statickým anebo dynamickým) a přesně jedním tvarem dynamickým, který k němu potřebujeme připojit. Nutno dodat, že ve stromové struktuře takových objektů musí být připojovaný objekt vždy potomkem tvaru, se kterým ho chceme spojit. Při vytváření vazeb mezi dvěma tvary je důležité dodržovat doporučené pokyny k docílení jejích správné funkcionality. Pokud je vazba vytvořená špatně, aplikace uživatele upozorní, že není v pořádku a vyzve ho, aby se na ní zaměřil a opravil ji. Následkem nekorektně vytvořeného spojení je totiž simulace, která není věrohodná a tedy taková, které jsme určitě nechtěli docílit.

4.7 Simulace

Uživateli jsou k ovládání simulace přístupné tři základní stavy, a to její běh, pozastavení a zastavení. Jejich aktivaci lze docílit pomocí vybrání odpovídající položky v menu aplikace anebo pomocí panelu nástrojů (tlačítka *start*, *pause* a *stop*). Interně však simulátor používá další stavy přechodné, všechny viditelné na obrázku 8, aby mohl správně informovat scripty anebo programy o tom, co bude právě následovat. K docílení správného chování by zmíněné scripty či programy měly vždy reagovat podle aktuální, systémem volané funkce, případně podle simulačního stavu. Osvědčeným postupem je správně rozvržené psaní kódu ve scriptech, jmenovitě rozdělení takového kódu do bloků v závislosti na tom, kdy se který blok má vykonat. Například pro *non-threaded child* script anebo *main* script základní rozvržení vypadá takto:

1. Inicializační část – kód v tomto bloku se vykonává pouze jednou, a to v momentě, kdy je script systémem inicializován.

2. Ovládací část – blok je volán v době aktivační fáze každého simulačního kroku.
3. Snímací část – kód tohoto bloku se vykonává v každém kroku simulace ve fázi jeho snímání.
4. Zotavovací část – blok je volán pouze jednou, v době, kdy dochází k zastavení simulace anebo když je script zahazován.



Obrázek 8: Stavy simulace [2]

4.7.1 Rychlost simulace

U simulací, které neoperují v reálném čase, je jejich rychlost závislá hlavně na dvou faktorech: simulačním kroku a počtu simulačních průchodů na jedno zobrazení. V opačném případě, tedy u simulací, které již počítají s reálným časem, závisí její rychlost hlavně na *real-time* multiplikačním koeficientu, ale rovněž určitém časovém kroku.

Robotický simulátor V-REP umožňuje rychlost simulace zvětšit, a to tak, že prvotní časový krok nastavený při spuštění zůstává stejný a zvyšuje se pouze počet simulačních průchodů potřebných k jeho zobrazení. Na druhou stranu, při zpomalování simulace pod inicializační hodnotu časového rámce a množství simulačních průchodů na jeden snímek, je tento krok již zmenšován.

4.7.2 Threaded rendering

Operace vykreslování vždy zvyšuje dobu trvání simulačního cyklu, jehož je součástí, čímž je souběžně zpomalovaná i simulace samotná. Jak již bylo řečeno, počet simulačních cyklů, tedy

množství opakování hlavního scriptu (*main* script), na jedno vykreslení lze definovat, nicméně ani to v některých případech nemusí stačit k přesnému běhu simulace, a to z toho důvodu, že vykreslování zpomalí každý *x-tý* simulační cyklus. Pro tyto případy aplikace nabízí možnost spustit speciální funkci, takzvaný *threaded rendering*.

Když je zmiňovaná funkce aktivní, pak se simulační cyklus skládá pouze z opakovaného provádění hlavního scriptu, a tedy simulace jako taková běží maximální rychlostí a není ničím degradována. Vykreslování scény v tomto případě probíhá pomocí separé vlákna. Bohužel se ale nejedná o funkci stoprocentně dokonalou, takže její použití sebou nese i řadu nevýhod, se kterými je třeba počítat, jako například: nesynchronní obraz vůči simulační smyčce, redukovanou stabilitu aplikace či v některých případech čekání na vykreslovací, respektive simulační vlákno (mezi jinými u odstraňování objektů). Ku příkladu u posledního zmíněného handicapu je klasické sekvenční vykreslování rychlejší než popisované renderování pomocí jiného vlákna, tudíž zůstává plně v rukou uživatele, jak si aplikaci přizpůsobí, a zdali se přikloní k přesné simulaci anebo raději upřednostní její věrné vylíčení.

4.8 Scripty

V-REP patří mezi vysoce přizpůsobitelné simulátory a téměř každý krok simulace je definován uživatelem. Této flexibility je docíleno integrovaným scriptovým interpretem.

Jazykem používaným pro psaní scriptů je Lua, což je rozšířený programovací jazyk určený k podpoře procedurálního, objektově orientovaného či funkcionálního programování. Lua kombinuje jednoduchou procedurální syntaxi s výkonnými konstrukcemi pro popis dat, které jsou založené na asociativních polích a rozšiřitelné sémantice. Je dynamicky typovaným jazykem s automatickou správou paměti, včetně *garbage collection*, což z něj činí ideální volbu pro psaní konfigurací či scriptů [6].

Program V-REP rozšiřuje základnu příkazů jazyka Lua, a přidává instrukce specifické jen pro něj, které lze rozpoznat podle prefixu „`sim`“.

V souvislosti se scripty používanými aplikací se bavíme o takzvaných vestavěných (*embedded*) scriptech. Vestavěný script je úzce spjat se scénou, respektive modelem, tedy jedná se o část kódu, která je jejich součástí a která bude ukládána a načítána společně s nimi. Existuje vícero typů scriptů, se kterými můžeme pracovat, a každý z nich obsahuje specifické funkce anebo se liší umístěním v rámci aplikace. Těmito typy jsou:

- Simulační scripty – scripty které jsou vykonávány pouze během simulace. Používají se k přizpůsobování simulace anebo simulovaných modelů. Hlavní simulační smyčka je zpracovávána prostřednictvím hlavního scriptu (*main* script), modely či roboti jsou zase kontrolováni pomocí scriptů označovaných jako *child*.

- Přizpůsobovací scripty – můžou být vykonány i když je zrovna simulace zastavena anebo pozastavena a jsou používány, jak už název napovídá, k přizpůsobování scény včetně všech objektů anebo simulace samotné.

Mimo tyto typy vestavěných scriptů se můžeme setkat se scripty, které nejsou přímo vnořené (neřadí se tedy mezi ty označované jako *embedded*), a které podporují přidávání zevnější funkcionality, a to takzvané *add-on* scripty.

Celou množinu scriptů můžeme dále neformálně rozdělit na přidružené a nepřidružené, kde do první kategorie zařadíme *child* a přizpůsobovací scripty, a do druhé hlavní script a *add-on* scripty. Přidružené scripty jsou vždy připojené k určitému objektu scény. Příkladem můžeme uvést model robota s přidruženým *child* scriptem. Pokud takového robota v rámci scény zkopírujeme, duplikujeme tím nejen samotný model, ale rovněž právě zmiňovaný script, který je odpovědný za jeho kontrolu při simulaci.

4.8.1 Main script

Jedná se o nepřidružený simulační script, tedy, jak už bylo vysvětleno výše, je vykonávaný pouze za běhu simulace a není přímo spjatý s objekty scény. Ve výchozím nastavení každá scéna obsahuje právě jeden hlavní script, jehož obsahem je základní kód, který umožňuje běh simulace. Bez takového scriptu by simulace jednoduše nefungovala. Obsahuje funkce (základní rozdělení kódu do bloků bylo již vypsáno v úvodu této podkapitoly), které jsou vhodně volány systémem, za předpokladu, že jsou v hlavním scriptu definované. Všechny funkce, kromě inicializační, jsou volitelné. Hlavní script není doporučeno modifikovat, a to například z důvodů, že:

- Výhodou simulátoru V-REP je, že jakýkoliv model (robot, senzor atd.) lze jednoduše vkopírovat do scény s jistotou toho, že bude okamžitě provozuschopný. Při úpravě hlavního scriptu uživatel riskuje to, že se modely nebudou chovat tak, jak se to u nich předpokládá. Dejme tomu, že uživatel modifikuje *main* script a při jeho obměně zapomene na instrukci `sim.handleChildScripts`, v tom případě, všechny modely, které se nachází ve scéně a jsou řízený právě přidruženými *child* scripty, nebudou fungovat.
- Udržováním originálního hlavního scriptu docílíme toho, že staré scény, vytvořené v předchozích verzích aplikace V-REP, budou kompatibilní i ve verzích novějších. Pokud tedy vývojáři přidají novou funkci, která je volána právě v hlavním scriptu, bude takovou funkcí automaticky aktualizován i *main* script staré scény.

Pokud uživatel z jakéhokoliv důvodu potřebuje hlavní script upravit, může ho otevřít pomocí příslušné ikony červeného scriptu v hierarchii scény. V momentě, kdy je script poprvé otevřen, označí se jako modifikovaný, a přestává být automatický aktualizován novými funkcemi.

4.8.2 Child script

Stejně jako *main* script, i *child* script (v rámci této diplomové práce dále označován i jako podřízený) patří mezi scripty simulační, nicméně v tomto případě už přidružené. Aplikace dovoluje

mít ve scéně neomezený počet takových scriptů, a každý z nich představuje sbírku rutin, zapsaných pomocí programovacího jazyka Lua, která umožňuje zacházení s určitou funkcionalitou simulace.

Tyto scripty jsou spojené s objekty scény, a v její hierarchii označené ikonou bílého, respektive modrého scriptu, jejichž pomocí lze otevřít editor s příslušným kódem. Pomocí odpovídajícího dialogu nastavení lze měnit konfiguraci jednotlivých scriptů či je spojovat s jinými objekty, které ještě s žádným scriptem spjaté nejsou. Takovým objektům lze asociovat i script nový, který je prázdný a čeká na úpravu ze strany uživatele.

Asociace podřízeného scriptu k objektu samotnému má v programu V-REP důležitou roli a přináší řadu výhod, jako například přenositelnost, kdy podřízené scripty jsou ukládány a načítány společně s objektem ke kterému náleží. S jejich pomocí lze vytvářet přenosný kód ale především modely, které nejsou na ničem závislé. Plně funkční model pak lze zabalit do jediného souboru. Dále můžeme zmínit individuální škálovatelnost, kterou lze specifikovat tak, že v momentě, kdy je v rámci jedné scény zkopírován model, který je asociován s podřízeným scriptem, duplikuje se jak model, tak i script samotný. I přesto, že zkopírovaný script bude identicky s originálem, bude fungovat korektně (například odkazovat na správné objekty v rámci modelu). Mezi další pozitiva provázání objektu se scriptem patří ku příkladu absence konfliktů mezi různými verzemi modelů či jednoduchá synchronizace se simulační smyčkou.

Skupinu asociativních, tedy přiřazených, podřízených scriptů můžeme rozdělit na dvě kategorie:

- *Non-threaded* podřízené scripty – obsahují kolekci blokujících funkcí, což znamená, že pokaždé, kdy je script zavolán, vykoná určitou funkci a pak vrátí řízení. Pokud není zmiňované řízení vráceno zpět, celá simulace se zastaví. Instrukce tohoto scriptu jsou volány scriptem hlavním, vždy dvakrát při každém kroku simulace, a to pomocí funkcí v jeho ovládacím a snímacím bloku. Non-threaded podřízené scripty dodržují přesně stanovený plán provádění příkazů, který ve výchozím nastavení začíná u volání scriptů přiřazených objektům potomků, směrem k předkovi. V hierarchii scény jsou označeny bílou barvou.

- *Threaded* podřízené scripty – fungují v odděleném vláknu. Zahájení procesu spuštění separé vlákna má ve výchozím nastavení na starosti *main* script a jeho přímo k tomuto účelu určené funkce. V porovnání s *non-threaded child* scripty můžou způsobovat určité problémy, které většinou vznikají jejích nekorektním programováním (například mrhání časem zpracovávání). V hierarchii scény *threaded child* scripty poznáme podle jejich modrého zabarvení.

4.8.3 Exekuční plán scriptů

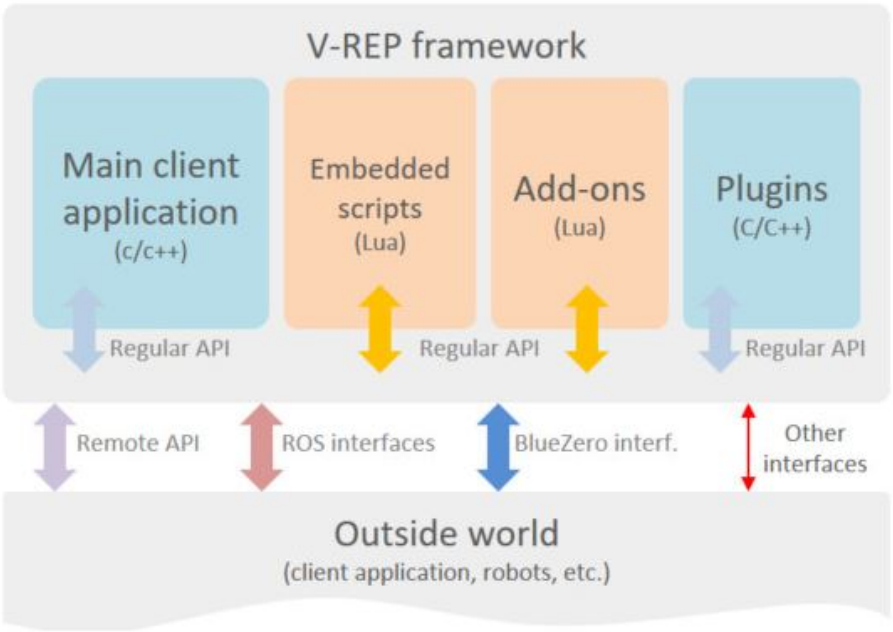
Scripty nejsou prováděny a volány náhodně. Typ scriptu, jeho umístění anebo nastavení ovlivňují, kdy dojde k jeho provedení. Exekuční plán na prvním místě zohledňuje to, o jaký typ scriptu

se jedná a od prvního k poslednímu jsou volány takto: *threaded child* scripty, *non-threaded child* scripty, přizpůsobovací scripty a *add-on* scripty.

Jelikož podřízené scripty patří do kategorie kódů simulačních, k jejích volání a vykonávání bude docházet pouze za předpokladu běžící simulace, a to znamená, že nejsou trvalé či perzistentní. Toto není případem přizpůsobovacích a *add-on* scriptů, které běží i když je zrovna simulace v pasivním stavu. K tomu nutno dodat, že *add-on* scripty tvoří kategorii zcela odlišnou a fungují neustále, například i u přepínání mezi různými scénami.

4.9 V-REP API framework

V-REP API framework seskupuje všechna rozhraní kolem programu a větví se do šesti kategorií. Jejich přehled je zobrazen na obrázku 9.



Obrázek 9: V-REP framework [2]

První ze zmiňovaných kategorií tvoří *regular* API, které je složeno z několika set funkcí, které lze volat z hlavní aplikace anebo vestavěných scriptů. Funkce a konstanty lze rozlišit podle prefixu „`sim`“. Existuje u něj možnost dalšího rozšiřování o speciální, uživatelem definované Lua funkce.

Další hlavní kategorii V-REP API frameworku tvoří *remote* API, které umožňuje komunikaci mezi programem a externí aplikací. Je multiplatformní, podporuje volání služeb a obousměrné streamování dat. Aktuálně se tato skupina dělí na dvě rozhraní: *BØ-based remote* API (přidáno ve verzi V-REP V3.6.0 vydané 15. února 2019) a původní *remote* API. První zmíněné reprezentuje vzdálené programovací rozhraní, které je založeno na BlueZero *middleware* a jeho pluginu. V porovnání s API první generace je flexibilnější, ale hlavně snadno rozšiřitelné. Podporuje pro-

gramovací, respektive skriptovací jazyky C++, Java, Python, Matlab a Lua. Na druhou stranu, původní API první generace obsahuje méně závislostí než jeho následovník, a navíc disponuje podporou programovacích jazyků C a Octave.

Mezi zbylé čtyři rozhraní, které tvoří V-REP API framework jsou řazeny: ROS a BlueZero, pomocné API a speciální kategorií tvoří jiná rozhraní, pomocí kterých můžeme dále framework rozšiřovat.

Zatímco k regulárnímu rozhraní přistupujeme přímo ze simulátoru, například pomocí vestavěného scriptu anebo hlavní aplikace, *remote* API, ROS a BlueZero rozhraní jsou přístupné pomocí téměř všech možných externích aplikací anebo hardwaru (reální roboti, vzdálené počítače atd.).

4.9.1 Legacy remote API

Názvem *legacy remote* API je označováno vzdálené rozhraní první generace. Jak již bylo řečeno, jedná se o součást V-REP API frameworku, která dovoluje kontrolovat simulaci, respektive simulátor, pomocí extérní aplikace anebo hardwaru, ať už se jedná o opravdového robota či jednoduše počítač.

Vzdálené API se skládá z přibližně jednoho sta specifických funkcí plus jedné funkce generické (pomocí které může uživatel vzdáleně volat instrukce simulačních scriptů), které lze provádět v C/C++ nebo Java aplikacích, Python anebo Lua scriptech či Matlab nebo Octave programech. Funkce rozhraní komunikují s programem V-REP pomocí socketové komunikace, která je před uživatelem skrytá a probíhá mezi dvěma separátními entitami, jmenovitě mezi klientem, tedy externí aplikací, a serverem, který reprezentuje simulátor V-REP.

Toto API by se za žádných okolností nemělo kombinovat s jeho následovníkem, vzdáleným rozhraním druhé generace (*B0-based remote* API), které oferuje větší flexibilitu, jednodušší použití a méně náročné rozšiřování funkcionality. V době tvorby této diplomové práce však API druhé generace ještě nebylo dostupné, tudíž jedinou možností jak vzdáleně přistupovat k programu V-REP a ovládat tak simulaci či simulátor, bylo právě *legacy remote* API.

5 Závodní okruh a model automobilu

5.1 Analýza

K docílení simulace řízení automobilu po vyznačené trase je potřeba vytvořit a nastavit vzájemnou interakci mezi modelem vozidla a závodním okruhem. Tato část diplomové práce je řešena v robotickém simulátoru V-REP.

Umělé vícevrstvé neuronové sítě, které jsou natrénovány řídit výše uvedený model a které tvoří studenti v rámci předmětu *Neuronové sítě*, mají své specifické vstupy. Získávání hodnot těchto vstupů je řešeno v simulátoru *Neural Racer*. Nicméně, jak se dá předpokládat, aplikace V-REP funguje na zcela odlišné bázi, něž program určený pouze k tomuto úkonu. Zajištění stejných dat musí být tedy vyřešeno jiným způsobem.

K základnímu pohybu modelu automobilu po předem vyznačené trase, a tedy snaha o jeho udržení na středovém pásu vozovky, jsou zapotřebí dva typy vstupů. Graficky jsou znázorněny na obrázku 10, a jmenovitě se jedná o:

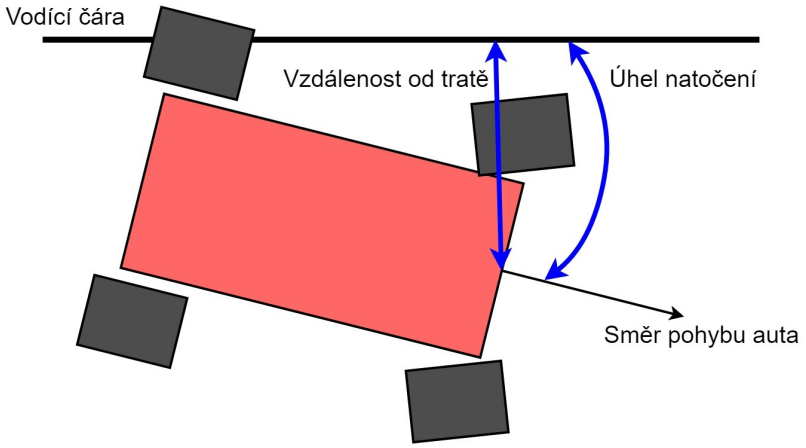
- Vzdálenost automobilu od vodící čáry – hodnota dosahující velikosti od 0 do 1. Krajní míry tohoto vstupu značí, že je automobil nalevo, respektive napravo od vodící čáry, a přitom se už nenachází na vozovce. Velikostí v rámci intervalu je určována vzdálenost vozidla od vodící čáry a jeho poloha vůči ní. Automobil se nachází uprostřed šíře vozovky (tedy přesně na vodící čáře) za předpokladu, že hodnota tohoto vstupu dosahuje velikosti 0,5. Neuronové sítě pro popisovanou vzdálenost obsahují pět vstupů, které se liší polohou vůči vozidlu samotnému (další hodnoty stanovují vzdálenost od vodící čáry za určitou časovou jednotku za předpokladu, že se automobil bude pohybovat stejným směrem).

- Úhel natočení vozidla vůči vodící čáře – hodnota rovněž nabývá velikosti 0 až 1. Tímto vstupem je neuronové sítí posílána informace o směru pohybu automobilu. Velikost 0,5 značí, že se vozidlo pohybuje přesně po směru trasy, kdežto krajní míry znamenají, že jede směrem opačným.

Další důležité hodnoty, které jsou potřeba na straně simulátoru získat, tvoří senzory. Pomocí nich lze neuronové síti zaslat informaci, že se v určitém okolí (před, za nebo u) automobilu nachází objekt, kterému se musí vyhnout. Ať už se jedná o jiný automobil či překážku na trati, senzor musí takovou skutečnost odhalit, a hodnotou v rozmezí od 0 do 1 oznámit, že se v jeho zorném poli něco vyskytuje. S rostoucí velikostí tohoto vstupu se snižuje vzdálenost detekovaného objektu od automobilu (hodnota 0 znamená že v poli senzoru se nenachází nic, s čím by mohlo dojít ke srážce, naopak velikost 1 značí, že je objekt v kolizní vzdálenosti od vozidla).

Pokud jsou neuronové sítí zaslány první dva typy vstupů, a síť samotná je naučená na ně reagovat, bude automobil úspěšně kopírovat předem vyznačenou trasu. Na druhou stranu, pokud jsou síti zaslány informace o tom, že se v poli senzorů vozidla nachází objekty, se kterými by mohlo dojít ke srážce, odpoví na ně nastavením řízení i za cenu dočasného vyjetí z tratě.

Ke správnému fungování simulátoru, respektive korektní interakci mezi simulátorem a neuronovou sítí, vzniká na straně serveru (aplikace V-REP) nutnost vytvořit model automobilu a trať. Trasa určuje, kudy by se měl model pohybovat, ten zas musí umět získávat o trati určitá data.



Obrázek 10: Základní parametry modelu vozidla

Data získané v simulátoru musejí být zaslány na stranu klienta (externí aplikace), kde jsou určitým způsobem zpracovány. Jedná se především o převod hodnot na takové, aby přesně odpovídaly vstupům neuronových sítí. I když lze většinu hodnot získat pomocí funkcí vzdálené API programu V-REP, některé funkce jsou podporovány pouze v *regular* API. Vzniká tedy potřeba takové data předzpracovat na straně simulátoru a zaslat je v podobě signálu, který bude dále upraven v externí aplikaci. Tohoto chování lze dosáhnout přizpůsobením *child* scriptů jednotlivých modelů.

5.2 Tvorba scény

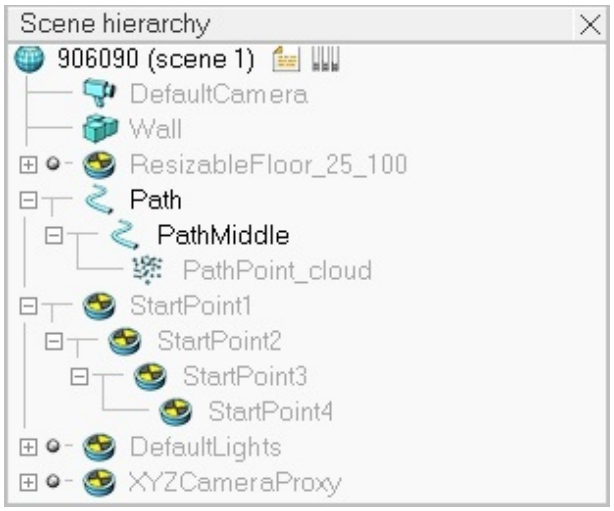
I když by se mohlo zdát, že ke korektní simulaci jízdy bude potřeba pouze správně vymodelované a nastavené vozidlo, není tomu tak. Konfigurace scény, ve které je uložená trasa tvoří polovinu úspěchu zmiňované simulace. Základní scéna se závodním okruhem, jejíž hierarchie je viditelná na obrázku 11 a výsledná podoba na obrázku 12, je postavená z těchto objektů: zdi, roviny, trasy a startovacích bodů.

5.2.1 Zeď

Zeď je jednoduchý tvar složený ze čtyř objektů sjednocených do jednoho. Účelem tohoto objektu je pouze ohraničit hrany roviny, aby nedocházelo k pádům vozidel mimo něj. Jedná se o nedetekovatelný, před kamerami skrytý objekt.

Pokud automobil řízeny neuronovou sítí vyjede z určené trasy, a shodou náhod se dostane do rohu roviny, může se stát, že se v takovém místě vlivem zdi zaklíní a nemůže pokračovat

dál. V tom okamžiku je nutný zásah uživatele, který musí zmíněný vůz vyprostit, ať už jeho natočením do jiného směru, anebo jeho celkovým přemístěním. Pokud bychom se chtěli této situaci vyvarovat, lze zdi jednoduše nastavit vlastnost *detecable*. Tím docílíme toho, že se jí bude model snažit vyhnout stejně jako jiným překážkám, které může na trati potkat. Je však potřeba počítat i s nevýhodami zmíněného řešení. Mezi jinými i s tím, že pokud trasa okruhu povede rovnoběžně s okrajem roviny a bude mu blízko, senzory určené primárně k detekci překážek před vozidlem mohou zaznamenat i zeď. Pak, i když automobil pojede přesně středem vodící čáry, může mít tendenci k uhýbání, a tedy objíždění zmiňované zdi.



Obrázek 11: Hierarchie scény s tratí

5.2.2 Rovina

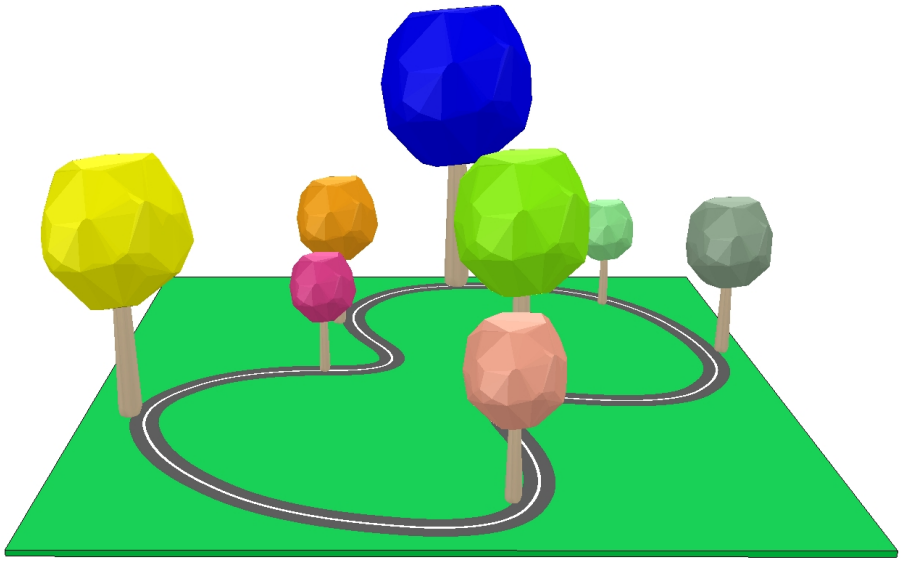
Rovina je tvořena jedním z výchozích objektů aplikace. Jedná se o model *resizable floor 25-100 meters.ttm*, který v sobě obsahuje přizpůsobovací script na změnu velikosti. Minimální délka strany 25 metrů je pro potřeby simulace automobilů dostačující (použité modely vozidel velikostně neodpovídají svým reálným vzorům, nevzniká proto nutnost tento objekt nějakým způsobem zvětšovat). Došlo tedy k odstranění přizpůsobovacího scriptu.

Původně byla rovina řešena jako několik menších objektů nakopírovaných vedle sebe, které tvořily celek o délce strany 25 metrů. Mezi jednotlivými prvky ale vznikaly viditelné spoje, tudíž došlo k jejich nahrazení jediným objektem roviny, což se nakonec ukázalo jako řešení jednodušší.

Objekt roviny je složen ze dvou elementů: *respondable* části, na které se můžou nacházet další objekty bez toho, aniž by ním propadly (za předpokladu že jsou k takovému chování nastaveny), a dále pak viditelným elementem s mřížkou. První zmiňovaný prvek viditelný není. K dosažení jednotného barevného provedení v rámci celé roviny (tedy k likvidaci mřížky která je v ní nativně obsažena), byl viditelný element odstraněn. U *respondable* prvku byla zapnuta viditelnost vůči základní kameře a přizpůsobena barva.

5.2.3 Trať

Dalším elementem scény je trať samotná. Jedná se objekt typu *path*, pomocí kterého lze v aplikaci V-REP v prostoru definovat trasu. Dělí se na dva typy: segmentový a kruhový. Pro potřeby závodního okruhu byl zvolen druhý uvedený. Objekty typu *path* samy o sobě nedělají nic, a ve většině případů jsou používány s objekty jinými. Stejně je tomu i v rámci scény se závodním okruhem, kde trasa slouží pouze k vizualizaci tratě, počítání úhlu natočení automobilu a směru jeho jízdy.



Obrázek 12: Závodní okruh

Při vytváření trasy se nativně zobrazí kruh s šestnácti body. Pomocí těchto bodů můžeme trasu upravovat podle svých představ a přizpůsobovat ji tak finálnímu tvaru, kterého chceme dosáhnout. Objekty cesty jsou ve scéně umístěny dva. Jeden tvoří vozovku v celé její šíři (*Path*), a druhý reprezentuje vodící čáru (*PathMiddle*). Jedná se o dva naprosto stejné objekty nakopírovány na sebe, které se liší pouze nastavenou šířkou a zvoleným barevným podáním. Bavíme se tedy o procesu modelování jediné trasy, a její následné duplikaci.

Trasa se vytvoří ve stejné výšce, jako rovina, což vede k jejich občasnému vzájemnému překrývání. Vzniká tedy nutnost objekt tratě nadzvednout o určitou hodnotu, aby k tomuto jevu nedocházelo. Vodící čára pak zase musí být umístěná výš než objekt reprezentující povrch vozovky. I když se jedná se o postup doporučený tvůrci simulátoru V-REP, a i přes to, že je nadzvednutí minimální, vzniká kvůli němu situace, kdy kola modelu vozidla působí dojmem, že jsou ve vozovce utopeny.

Jelikož jedním ze vstupů neuronových sítí je vzdálenost od tratě, nabízí se možnost využití výpočetního modulu minimální vzdálenosti mezi dvěma měřitelnými entitami. Objekty typu *path* ale bohužel nesplňují podmínku nutnou k provedení zmiňovaného měření. V jejich konfi-

guraci totiž nelze nastavit vlastnost *measurable*, která je pro využití funkce počítání minimální vzdálenosti nutná. Z toho důvodu bylo potřeba vytvořit další objekt, a to *PathPoint_cloud*.

Pomocí *point cloud* (typ objektu) můžeme vytvořit množinu bodů, která bude tvarově odpovídat vzorovému objektu typu *shape*. K vytvoření takové množiny je potřeba nejprve vygenerovat tvar, který by kopíroval vodící čáru. Z důvodů přesného měření byla zmiňovaná vodící čárá před vytvářením nového objektu zúžena, aby přesně odpovídala pouze středu vozovky. Pomocí dialogu nastavení tratě byl posléze vytvořen samotný tvar, který byl použit k vygenerování množiny bodů. Objekt typu *point cloud* už měřitelný je, tudíž k zjišťování vzdálenosti mezi středem vozovky (vodící čárou, která je reprezentována body) lze použit zabudovaný modul k tomu určený.

Tvar který posloužil jako vzor při vytváření objektu *PathPoint_cloud* už není k ničemu potřebný, tudíž je ze scény s okruhem odstraněn. Mezi jednotlivými objekty, které jako celek reprezentují trasu, po které se má vozidlo pohybovat, jsou vytvořeny vztahy předek-potomek. Vozovka tvoří úroveň nejvyšší, následuje vodící čára a s ní spojená množina bodů. Jelikož vizualizace měřitelné vodící čáry uživateli není k ničemu užitečná, je její zobrazení vypnuto.

5.2.4 Startovací body

Startovací body jsou reprezentovány čtyřmi objekty typu *dummy*. Tyto objekty jsou nejjednodušším typem objektu v aplikaci vůbec. Jedná se o bod scény, který má svoji pozici a orientaci. *Dummy* objekty jsou takzvané pomocné entity, které samy o sobě příliš užitku nepřináší.

Pomocí startovacích bodů jsou na trať umisťovány vozidla. Každý ze zmiňovaných bodů má přesně nastavenou pozici a orientaci. Nachází se v takové výšce nad tratí, že v momentě, kdy je na jejich místo přenesen automobil, stojí koly na ploše. Pomocí jejich orientace je zase umísťované vozidlo natočeno přesně ve směru jízdy daného závodního okruhu a pozice v rámci něj.

I když simulátor V-REP umožňuje grafické zobrazení objektů typu *dummy*, ve scéně s tratí jsou tyto body nastavený jako neviditelné. Rovněž je mezi nimi vytvořený vztah předek-potomek, což usnadňuje jejich eventuální přemisťování.

5.3 Modelování automobilu

Další důležitou části při simulování pohybu vozidla po vyznačené trase je automobil samotný. Takový vůz ve zmíněné simulaci plní dvě důležité role. Za prvé slouží coby vizualizace, pomocí které může člověk snadno ale hlavně hned vidět, jak se neuronové sítí, tedy řidiči automobilu, vede reagovat na trať včetně překážek na ní umístěných.

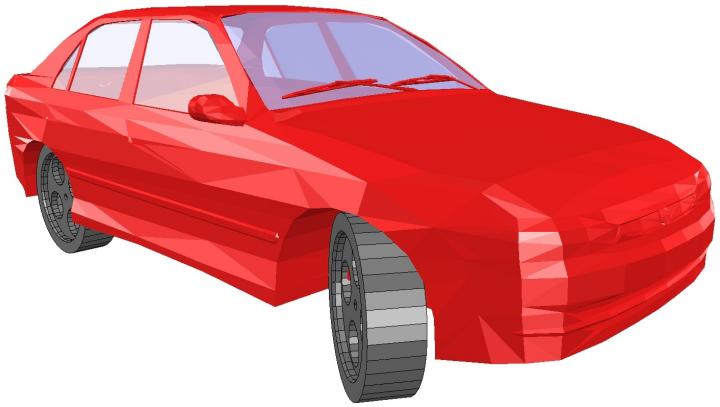
Je sice zřejmé, že k simulaci samotné její vizualizaci nepotřebujeme. V našem případě by ale taková simulace příliš užitku nepřinesla. Z numerických výstupů lze sice vyčíst ledaco, ale snažit se je pochopit v reálném čase je pro člověka úkol dosti složitý.

Druhou hlavní funkcí automobilu je prozkoumávání, tedy detekování či analýza, jeho okolí. Pomocí simulátoru, potřebujeme získávat určité hodnoty, které slouží jako impulsy pro neuronové

sítě. Podle nich pak zmiňované sítě odpovídají, a tudíž diktují vozidlu, co má dělat. Ať už se jedná o zatáčení koly, zpomalování či zrychlování atd.

Z výše uvedených informací lze vydedukovat, že model bude složen ze dvou částí. Viditelné, která poslouží k jeho zobrazení, a části různých senzorů a měřitelných objektů, které budou s modelem přímo spojeny a budou odpovědné za průzkum okolí. Druhá uvedená část, dále detekční, není při znázornění jízdy potřeba a je tedy před kamerami simulátoru skrytá.

Jak už bylo řečeno, viditelná část automobilu slouží ke grafickému znázornění jeho pohybu (tedy jízdy) po vyznačené trase. Finální model automobilu, jež pak uživateli slouží k simulaci řízení pomocí neuronových sítí, je zobrazen na obrázku 13.



Obrázek 13: Model vozidla

Model vozidla je složen z několika oddílů, které jsou mezi sebou vzájemně provázány a tvoří tak jeden celek. Takový model (bez detekční části) můžeme neformálně rozdělit do těchto základních segmentů:

1. Dynamický segment – pomocí kterého je simulovaná dynamika vozu.
2. Statický segment – který tvoří, ale hlavně vizualizuje karosérii a kola automobilu.
3. Dynamické spoje – které drží vůz pohromadě a tvoří z něj jeden celek.

Dynamický segment a spoje jsou rovněž skryté před kamerou simulátoru. Jejich role je ale u viditelné části vozu stěžejní, tudíž jsou uvedeny jako její součást.

5.3.1 Dynamický segment

Základ automobilu tvoří dynamický segment. Jedná se o jednoduchý kvádr, kterému lze v simulátoru nastavovat fyzické vlastnosti, jako je například hmotnost. Vychází z modelu *simple Ackermann steering.ttm*, který je součástí programu V-REP. Jelikož zmíněný model velikostně odpovídá reálnému vozu, byl pro zjednodušení zmenšen pomocí funkce škálování. Pro potřeby simulace více automobilů najednou, byly z jeho základu odstraněny tlumiče. Jak se ukázalo,

při současné jízdě vícera vozů i zanedbatelné dynamické simulování odpružení mělo za následek snížení výkonů celé simulace, a tedy její pomalejší běh.

Na zmíněný dynamický kvádr jsou napojeny čtyři, rovněž dynamické, válce. Ty plní roli kol modelu. Jsou odpovědné za jeho pohyb a změny směru. Zároveň drží základnu automobilu nad zemí a tvoří tak čtyři kontaktní body s rovinou.

Jak základna, tak i všechna kola jsou *respondable*, a reagují tedy na jiné objekty (modely) se stejnou vlastností. Díky tomuto nastavení do sebe můžou modely automobilů narážet mezi sebou. Zmiňovanou vlastností je rovněž docíleno toho, že automobil stojí pevně na rovině. Nutno zmínit, že u kol je nastavená jiná *respondable* maska, než je tomu u základny. Deficitem tohoto upřesnění by docházelo k interakci mezi samotnou základnou automobilu a koly k ní připojenými. U téže základny automobilu je navíc nastavená vlastnost *detecable*, což umožňuje jiným vozidlům na trati (respektive senzorům těchto vozidel) tento objekt snímat.

5.3.2 Dynamické spoje

Kola jsou k základně automobilu připojená pomocí dynamických spojů. U zadní nápravy se jedná o dva jednoduché spoje, pomocí kterých je nastavený volný pohyb napojeného dynamického tělesa. Zadní kola tedy reagují na pohyb otáčením, nicméně nemají na něj žádný vliv. U předního páru kol je situace poměrně jiná. Kromě dvou dynamických válců, které reprezentují kola samotná, jsou u nich umístěné ještě dva další, rozměrově menší, rovněž dynamické ale už bez vlastnosti *respondable*. Na každé kolo (složené z dvou neviditelných válců) pak připadají dva dynamické spoje, které fungují jako řízení, respektive motory. Natáčení kol vozu doleva či doprava je docíleno v hierarchii vyšším spojem (motorem). Jelikož simulátor V-REP neumožňuje napojení dynamického spoje (v módu umožňujícím pracovat jako motor) na jiný takový spoj, je k tomu potřeba mezičlánek. Zmiňovaným kusem je válec popisován výše. Na ten je posléze napojený další motor, který už funguje jako zdroj otáčení. Nejníže v takové hierarchii se pak nachází samotné *respondable* kolo. Užitím dvou dynamických spojů, které jsou ve funkci motorů, tedy docílíme toho, že se může kolo otáčet kolem jedné ze svých os, a rovněž natáčet kolem osy druhé.

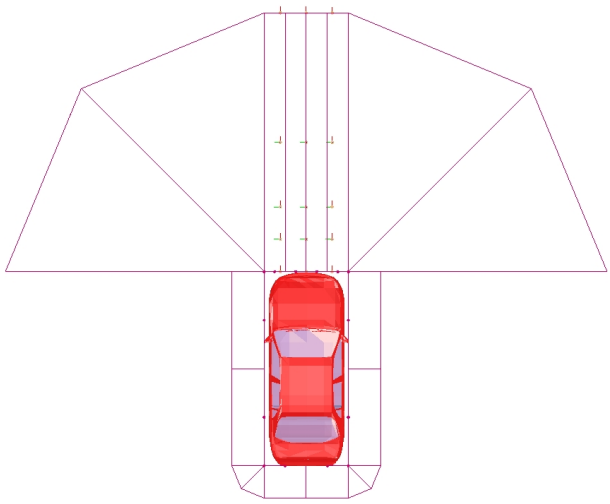
5.3.3 Statický segment

Automobil v doposud popsaném stádiu už je schopný jízdy. Nicméně o zařazení do kategorie vozidel podle jeho vzhledu, by se dalo jen diskutovat. Aby takový model jako automobil i vypadal, byla mu přidána karosérie (včetně oken) a věrnější tvary kol. Tyto objekty tvoří třetí, výše zmiňovaný statický segment celého modelu. Jedná se o komplexnější tvary, které jsou nastaveny jako statické a *non-respondable*. S nadřazeným objektem vždy tvoří vztah předek-potomek, a drží si proto přesně vůči němu vymezenou pozici po celou dobu běhu simulace.

5.4 Skupina detekčních objektů

Na rozdíl od části modelu určené primárně k vizualizaci a simulaci dynamiky, jeho druhý hlavní díl, tedy detekční skupina objektů, slouží k získávání dat o okolí vozu. Neuronové sítě využívané ve stejnojmenném předmětu umějí aktuálně reagovat především na tvar trasy a překážky. Jedná se o vstupy vzdáleností od tratě, úhlu natočení vůči ní a pak o data získané ze sítě senzorů přiblížení. U modelu automobilu je tedy vytvořen systém detektorů a snímačů, který toto umožňuje.

Detektory a snímače jsou u modelu automobilu skryté. Jednoduchým nastavením jejich viditelnosti vůči hlavní kameře lze ale takové nastavení změnit. Vozidlo s povolenou vizualizací zmiňovaných senzorů je zobrazeno na obrázku 14.



Obrázek 14: Model vozidla s viditelnou sítí senzorů

Sada entit, která je použitá ke snímání informací o okolí vozidla, je neformálně rozdělená do tří částí. Konkrétně na: objekty určování orientace, objekty určování vzdálenosti od tratě a senzory přiblížení k detekci jiných objektů. Všechny vyjmenované sady mají ve stromové struktuře modelu jeden nadřazený *dummy* objekt s názvem *detectors*. Ten je v roli jejich předka a sám je potomkem kostry vozu. Ke změně pozice (například vertikální) pak stačí pouze přizpůsobit umístění zmíněného nadřazeného objektu.

5.4.1 Orientace

Kolekci objektů k určování orientace, respektive natočení vozidla od tratě, tvoří dva *dummy* objekty. Jak již bylo jednou zmíněno, *dummy* objekt je prostý bod, který má svojí pozici a orientaci. Druhá vypsaná vlastnost tedy přímo vybízí k tomu, použít objekt tohoto typu právě k určování úhlu natočení automobilu od trasy.

Jedná se o objekty *orientationBackward* a *orientationForward*. První vyjmenovaný je opačně orientovaný vůči druhému, který kopíruje natočení vozidla v prostoru. Oba objekty jsou ve

vztahu předek-potomek s dalším *dummy* objektem *orientations*, který slouží pouze ke zjednodušení hierarchie modelu.

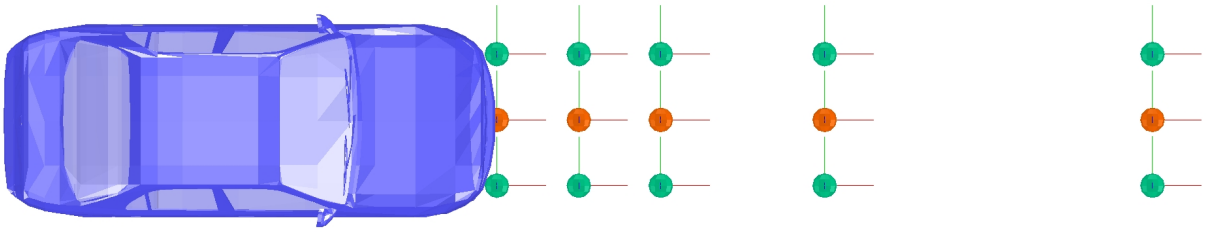
5.4.2 Vzdálenost od tratě

V podkapitole o tvorbě scény bylo vylíčeno, že středová čára tratě je přizpůsobená tak, aby byla měřitelná, respektive aby se dala měřit vzdálenost od ní. Ve skupině detekčních objektů modelu vozu jsou proto zařazeny takové objekty, od kterých se vzdálenost právě ke středové čáře počítá.

Jedná se opět o *dummy* objekty. Mezi další základní funkce těchto objektů totiž patří, že jim lze natavit vlastnost *measurable*.

Základní oddíl pro získávání vzdálenosti tvoří tři takovéto objekty. Jeden z nich, centrální, je umístěn na pomyslné čáře určující střed vozidla, těsně před jeho dynamickou (a rovněž i *respondable*) částí. Je označen jako *distance00*. S ním jsou vztahem předek-potomek spojeny zbylé dva. Jeden z nich, *distance00Left*, je polohován na levou stranu od něj, druhý, označený jako *distance00Right* zas na stranu opačnou. Pomocí centrálního *dummy* objektu je počítána samotná vzdálenost od tratě. Další dva jsou použity pro upřesnění, zdali se zmiňovaný centrální prvek nachází na levé, respektive na pravé straně od vodící čáry.

Neuronové sítě vyžadují na vstupu pět hodnot popisujících vzdálenost automobilu od tratě, kde každá je v určitém odstupu od vozidla. Proto model obsahuje stejné množství výše popisovaných sad. Lze je rozeznat podle čísla uvedeného v jejich názvu, které symbolizuje relativní vzdálenost, přesněji poměr takových vzdáleností od zmíněného modelu automobilu. Všechny objekty typu *dummy* využívané k výpočtu vzdálenosti vozidla od vodící čáry jsou viditelné na obrázku 15.



Obrázek 15: Objekty výpočtu vzdálenosti od tratě

Stejně jako u objektů určených ke zjišťování úhlů natočení vozidla, i tato sada objektů je potomkem společného předka *distances*, který zjednodušuje hierarchii modelu.

5.4.3 Detekce objektů

Poslední kategorií detekční skupiny jsou senzory přiblížení. Pomocí nich lze zjistit, zdali se v určité poloze před, za nebo kolem auta nenachází detekovatelné objekty. Neuronové sítě, které řídí vozidlo, mají za účelem detekce předmětů dohromady 18 vstupů. Nejpočetnější část tvoří vstupy, určené k odhalování objektů před automobilem. Je jich 8. Po stranách jsou rozmístěné

dohromady 4 takové senzory, na každou ze stran rovným dílem. Zbylých 6 se nachází vzadu za vozidlem. Jsou tvořeny objekty typu *proximity sensor*. Aplikace V-REP nabízí velmi výkonný a efektivní mechanizmus simulace takových senzorů.

Hierarchii začíná pro všechny senzory stejný předek, objekt typu *dummy* s názvem *sensors*, jehož přímými potomky jsou čtyři objekty stejného typu. Jmenovitě se jedná o: *sensorsFront*, *sensorsRight*, *sensorsLeft* a *sensorsRear*. Tyto objekty slouží pouze k snadnější orientaci mezi jednotlivými senzory, tedy k jejich kategorizaci.

Potomky výše uvedených pomocných objektů jsou už samotné senzory přiblížení. Ty jsou rozděleny do bloků, kde každý zmiňovaný blok odpovídá jednomu vstupu neuronové sítě. Jejich pojmenování je identické.

V původním simulátoru *Neural Racer* byla vzdálenost detekovaného objektu od vozidla v rámci jednoho bloku určovaná způsobem odlišným, než je tomu aktuálně v programu V-REP. Jednotlivé bloky byly rozděleny do dalších deseti oddílů. Snímání v rámci zmiňovaných oddílů pak určovalo vzdálenost automobilu od překážky na trase. Například tedy, pokud detekoval překážku pátý oddíl nějakého bloku, znamenalo to, že se taková překážka nachází přibližně v polovině snímané délky, čemuž odpovídala i konstantní velikost zasílaná na vstup neuronové sítí.

Po vzoru tohoto modelu byla v prvotní fázi vytvořená síť senzorů i v simulátoru V-REP. Toto řešení se ovšem ukázalo jako nešťastné. Model automobilu obsahoval 180 snímačů, a simulace jen jednoho (modelu) z nich byla výpočetně tak náročná, že souběžná interakce vícera vozů v rámci jedné scény nepřicházela v úvahu. Zmiňované oddíly byly proto sjednoceny a nahrazeny jenom jedním objektem typu *proximity sensor*. Aplikace V-REP totiž nativně umožňuje u každého senzoru přiblížení zjistit, jak daleko se nachází detekovaný objekt od jeho snímacího bodu.

5.5 Script modelu

Jednu z nejdůležitějších části modelu tvoří jeho podřízený script. Primárně jsou takové *child* scripty určeny k samotnému ovládání modelu, jemuž jsou přiřazeny. K simulaci řízení jsou v tomto případě použity neuronové sítě, tudíž by se nabízela možnost, vestavěný script na straně serveru (tedy v aplikaci V-REP) vynechat úplně. Takový postup ale dodržet nelze. Důvodem je absence některých stěžejních funkcí v *remote* API, které jsou potřebné k získání všech potřebných hodnot, respektive vstupů neuronových sítí. Podřízený script u modelu automobilu tedy neplní funkci ovladače (*controlleru*) samotného, nýbrž pomáhá zprostředkovat určité veličiny, které jsou k jeho řízení nezbytné.

5.5.1 Inicializační část scriptu

Podle doporučených postupů je podřízený script modelu rozdělen do segmentů. První z nich tvoří inicializační část. Její základním účelem je získat informace o objektech ve scéně, včetně samotného modelu, se kterým je script spjatý.

Inicializační část scriptu je od jeho zbylé částí rozlišena kladným vyhodnocením podmínky `sim_call_type==sim.syscb_init`. Tímto se jednoznačně určí, že se jedná o část kódu, která se vykoná pouze jednou v rámci celé simulace. V dalších krocích zmiňované simulace už je uvedená podmínka vyhodnocená negativně, tudíž se kód jí podřízený nebude opakovat. Toto chování je žádané, jelikož informace o objektech (jejích takzvané *handles*), se v rámci běhu simulace nemění.

Důležitou části inicializačního segmentu podřízeného scriptu je přesná specifikace objektů, které je docíleno pomocí funkce `sim.getNameSuffix`. Pokud jsou v rámci jedné scény umístěné dva naprosto identické modely, program V-REP je musí umět určitým způsobem rozlišit. Toho je docíleno přidáváním znaku # a číselného suffixu. Aby tedy bylo možné simulovat řízení více stejných modelů vozidel, u načítání jeho jednotlivých součástí (například objektu *distace00* viditelného ve výpisu 1), je potřeba přesně specifikovat jeho název, včetně zmiňovaného suffixu. Za předpokladu, že se ve scéně nachází pouze jeden automobil, popisovaný krok není potřeba.

Tímto způsobem upřesněné názvy objektů jsou následně použity k získání jejich číselné reprezentace v rámci scény. Pomocí funkce `sim.getObjectHandle` s parametrem názvu objektu je takové číslo načteno a uloženo k dalšímu zpracování v rámci simulačního scriptu (ať už do pole, které reprezentuje množinu objektů používaných k získávání informací o trati, anebo pouze do proměnné).

```
if (sim_call_type==sim.syscb_init) then
    nameSuffix=sim.getNameSuffix(nil)
    if nameSuffix>=0 then
        distance0String='distance00#'..nameSuffix
    else
        distance0String='distance00'
    end
    distanceArray={sim.getObjectHandle(distance0String)}
    pointCloud=sim.getObjectHandle('PathPoint_cloud#')
    pathMiddle=sim.getObjectHandle('PathMiddle#')
end
```

Výpis 1: Inicializační část scriptu

5.5.2 Snímací část scriptu

Na rozdíl od inicializačního segmentu, snímací blok podřízeného scriptu je vykonán v každém kroku simulace. Pomocí vyhodnocení podmínky `sim_call_type==sim.syscb_sensing` se rozlišuje od zmiňované inicializační části scriptu. Pracuje s objekty, jejichž informace (tedy číselná reprezentace v rámci scény), byly získány právě v bloku inicializačním.

Snímací blok *child* scriptu využívá řadu různých funkcí programu V-REP, pomocí kterých je zjišťována například minimální vzdálenost mezi objekty, pozice v rámci objektu typu *path*,

orientace objektů či jejich nastavení. Jedná se, jak už bylo psáno výše, o funkce, které nejsou dostupné pomocí externí aplikace, respektive pomocí *remote* API. Všechny tyto informace jsou následně zabaleny do signálů, které jsou specifikované názvem a které hodnoty přenesou coby prostý text. Takový signál je posléze možné v externí aplikaci přijmout, rozložit na jednotlivé číselné části a předat je k další analýze.

Příklad získávání hodnot vzdálenosti od tratě je uveden ve výpisu 2. Pro všech pět objektů určených k její měření je funkcí `sim.checkDistance` získána minimální délka mezi zmiňovaným objektem a středem vodící čáry, kterou představuje množina bodů (objekt typu *point cloud*). Všechny výsledné hodnoty funkce, které reprezentují jak vzdálenost, tak přesné souřadnice bodů, mezi kterými se nachází, jsou následně zabaleny do jednoho řetězce pomocí funkce `sim.packFloatTable`. V posledním kroku následuje nastavení samotného signálu, čehož je docíleno funkcí `sim.setStringSignal`.

```
for i=1,5,1 do
    local res,d=sim.checkDistance(distanceArray[i],pointCloud,0)
    local seg={d[1],d[2],d[3],d[4],d[5],d[6],d[7]}
    local data=sim.packFloatTable(seg)
    if nameSuffix>=0 then
        sim.setStringSignal('distance'..i..'#'..nameSuffix,data)
    else
        sim.setStringSignal('distance'..i,data)
    end
end
```

Výpis 2: Snímací část scriptu

Obdobným způsobem, jak bylo znázorněno na příkladu získávání vzdálenosti automobilu od vodící čáry, jsou na straně simulátoru předzpracovány i zbylé hodnoty, které pomocí *remote* rozhraní získat nelze. Například poloha v rámci délky tratě, typ senzorem snímaného objektu atd.

6 Implementace klienta

6.1 Analýza

K úspěšnému napojení neuronových sítí, coby pomyslných řidičů modelů vozidel v simulátoru, vzniká potřeba implementace programu, který umí zajistit komunikaci mezi aplikací V-REP a samotnou naučenou neuronovou sítí. Jelikož aplikace V-REP disponuje řadou API, které v sobě zapouzdřují základní funkce k ovládání simulace, respektive simulátoru, stává se takový úkol *jednoduše* řešitelným.

Na začátku tvorby programu ovšem vzniká jedna důležitá otázka. Protože *remote* API simulátoru podporuje více programovacích jazyků, je potřeba se hned u zrodu aplikace rozhodnout, jakým směrem se vydat a pomocí kterého z nich výsledné řešení implementovat. Je zřejmé, že takové rozhodnutí je zásadní, a jeho následná změna bude spíše nemožná, anebo minimálně těžce realizovatelná. Volba nakonec padla a výsledné řešení programu je psáno programovacím jazykem Java (vývojové prostředí Eclipse, verze 2018-12). Hlavním důvodem tohoto rozhodnutí byla nižší znalost jiných, v rámci *remote* API dostupných jazyků.

Jelikož jde o program, který plní roli hlavního zprostředkovatele mezi neuronovou sítí a simulátorem, měl by splňovat základní požadavky. Mezi ně jsou zařazeny například úspěšné napojení zmiňované neuronové sítě na model simulovaného vozu, korektní zpracování dat přijatých ze simulátoru anebo odeslání výsledků, které poslouží coby impulsy k řízení.

6.2 Architektura

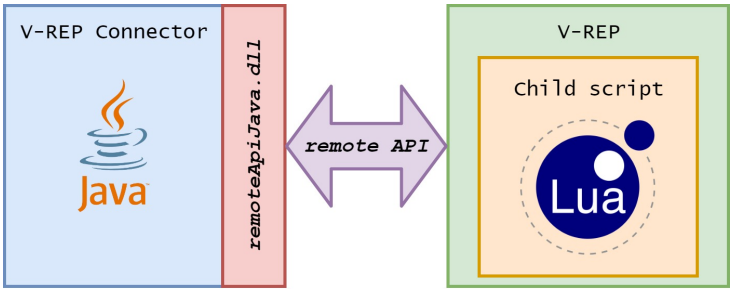
Při vytváření aplikací je kladen veliký důraz na její architekturu. Implementovat program totiž nutně neznamená vytvořit pouze jediný projekt. Aktuálně jsou vývojáři využívaný různé vzory definovaných architektur (například klient-server, třívrstvé rozdělení atd.). Takové architektury většinou definují vrstvy, do kterých je aplikace jako celek rozdělená. Každá ze zmiňovaných vrstev je pak odpovědná za specifickou funkcionalitu. Komunikací mezi těmito vrstvami je docíleno funkčnosti programu jako jednoho celku. K rozdělení dochází zpravidla v situacích, kdy každá z vrstev funguje na odlišné infrastruktuře, což je i příklad komunikace mezi externí aplikací a programem V-REP, graficky prezentován na obrázku 16. O architektuře se bavíme rovněž v případě, kdy je celá aplikace rozdělená do logických bloků, už jen z důvodů její jednodušší strukturalizace.

Ke zmiňovanému logickému rozdělení do vrstev došlo i u vytváření aplikace k řízení vozidel neuronovou sítí, dále také označované názvem *V-REP Connector*. Program je složen ze tří základních úrovní, kde každá plní svojí specifickou roli. Celek tak tvoří:

1. Prezentační vrstva – obsahuje třídy programu zodpovědné za jeho grafické znázornění. Je seskupena v balíčku *gui*. Všechny třídy v ní zakomponované lze rozeznat podle prefixu „*Gui*“.

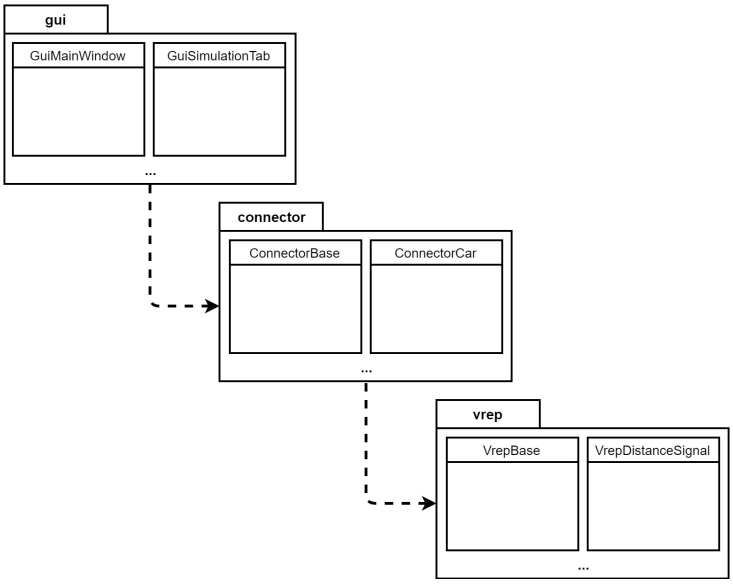
2. Logická vrstva – tvoří stěžejní část programu. Obsahuje třídy odpovědné za správné fungování simulace, obsluhu a zpracovávání dat, včetně implementace načítání a dotazování neuronových sítí. Je reprezentována balíčkem s názvem *connector* a všechny třídy, které mu náleží nesou stejný prefix začínající velkým písmenem.

3. Komunikační vrstva – zapouzdřuje v sobě základní příkazy *remote* API programu V-REP, které jsou potřebné k ovládání simulace z extérní aplikace. Je seskupena v balíčku *vrep*. Všechny třídy, které obsahuje zmiňovaný balíček, jsou rozeznatelné prefixem „*Vrep*“.



Obrázek 16: Architektura

Komunikace mezi uvedenými vrstvami programu je zprostředkovaná instancemi tříd nižší vrstvy ve vrstvě jí nadřazené. Základní znázornění (z důvodu čitelnosti jsou v každém balíčku uvedeny pouze dvě třídy) architektury pomocí *package* diagramu jazyka UML je možné vidět na obrázku 17.



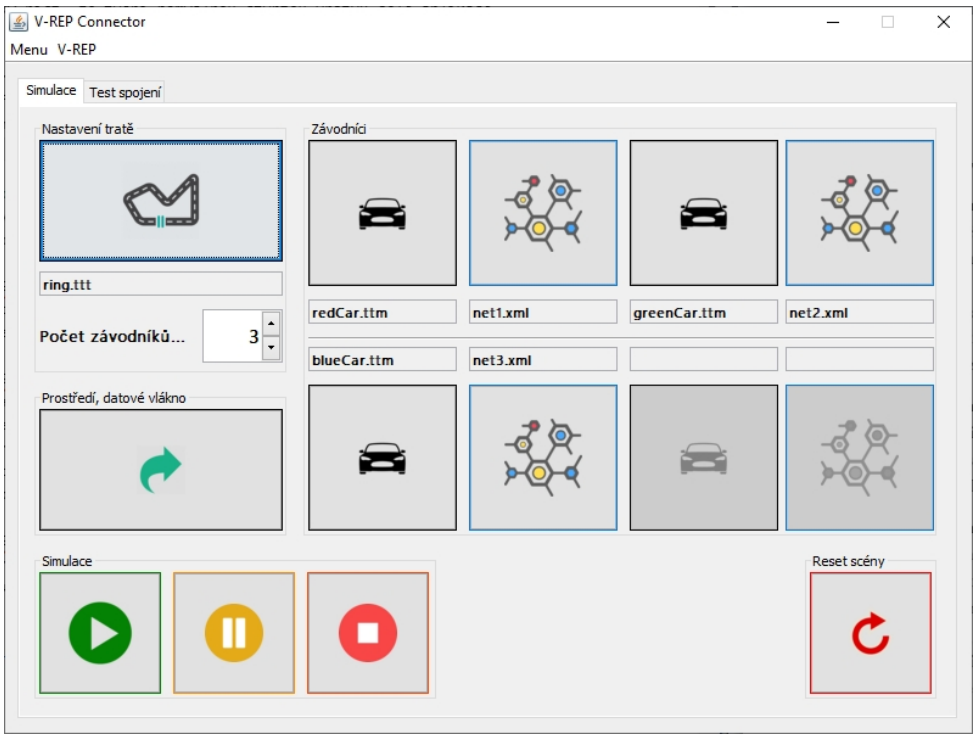
Obrázek 17: Package diagram aplikace

Kromě výše uvedených balíčků, aplikace obsahuje ještě jeden. Ten už ale s její základní vrstvenou architekturou nikterak nesouvisí. Jedná se o sadu pomocných tříd přímo určených pro

práci se simulátorem V-REP. Zmiňované třídy jsou seskupeny v balíčku *coppelia*. Nutno dodat, že jednou z těchto tříd je i `remoteApi`. Ta je odpovědná za načítání knihovny *remoteApiJava.dll*, která obsahuje implementaci všech externě dostupných metod programu V-REP. Instance této třídy je vytvářena v komunikační vrstvě, dalo by se tedy říct, že tvoří pomyslnou čtvrtou vrstvu celé aplikace.

6.3 Uživatelské rozhraní

K implementaci uživatelského rozhraní byl použit volně dostupný nástroj pro vývojové prostředí Eclipse – WindowBuilder [7]. Samotné grafické rozložení je pak tvořeno pomocí knihovny uživatelských prvků Swing [8]. Finální vzhled hlavního panelu programu, který slouží k ovládání simulátoru, je zobrazen na obrázku 18.



Obrázek 18: Uživatelské rozhraní aplikace

Uživatelské rozhraní programu *V-REP Connector* se vytváří ve třídě `GuiMainWindow` a jedná se o základní okno (instance třídy `JPanel`) s pevně nastavenou velikostí. K tomu jsou dále připojeny jednotlivé prvky, které slouží k ovládání programu či simulace, kterými jsou:

- Instance třídy `GuiMenu` – tvořená objektem `JMenu`. Vykresluje základní menu aplikace. To uživateli poskytuje možnost vypnutí programu, zobrazení informačního okna anebo spuštění aplikace V-REP. Dále obsahuje reference na web *Coppelia Robotics*, online manuál simulátoru V-REP a odkaz k jeho stažení.

- Instance třídy `GuiSimulationTab` – tvořená objektem `JPanel`. Tato třída v sobě obsahuje hlavní skupinu ovládacích prvků k řízení simulátoru. Pomocí nich uživateli poskytuje možnost výběru závodního okruhu (předem uložené scény) a počtu závodníků, kteří se na dané trase později objeví. Ke každému závodníkovi je přiřazován vybraný model vozu, který je párován se zvolenou neuronovou sítí. Po úspěšném nastavení všech potřebných vstupů pak uživateli dovoluje načíst vybranou trasu v programu V-REP. V tomto momentě je už simulace připravená k běhu, a proto jsou součástí třídy prvky k její ovládání anebo případnému kompletnímu resetu.

- Instance třídy `GuiTestTab` – tvořená rovněž objektem `JPanel`. Tato třída v sobě zapouzdřuje sadu ovládacích prvků k snadnému otestování všech stěžejních funkcí, které jsou ke korektnímu běhu programu, respektive simulace v aplikaci V-REP, potřeba.

Instance tříd `GuiSimulationTab` a `GuiTestTab` jsou do hlavního okna aplikace přidány pomocí objektu `JTabbedPane`.

Kromě výše uvedených tříd jsou v prezentační vrstvě uloženy ještě dvě pomocné: `GuiHelper` a `GuiTestLayout`. První uvedená obsahuje sadu pomocných funkcí využívaných v celé šíři nejvyšší vrstvy aplikace, druhá je odpovědná za změny rozhrání v rámci třídy `GuiTestTab`.

Na všech ovládacích prvcích, jejichž většina je tvořená objekty `JButton`, jsou registrování posluchači, pomocí kterých je docíleno reakcí jednotlivých, uživatelem žádaných podnětů.

6.3.1 Nahrávání scény

Jak už bylo řečeno výše, třída `GuiSimulationTab` v sobě obsahuje hlavní skupinu ovládacích prvků k řízení programu a simulace. Vytváří se v ní instance třídy `ConnectorSimulation`, čímž je docíleno spojení mezi prezentační a logickou vrstvou programu. Dále obsahuje pomocné funkce k registrování vozidel a neuronových sítí včetně kontroly jejich výběru.

Nejdůležitější část této třídy tvoří posluchač nastavený na tlačítko `btnSceneLoad`, zobrazený ve výpisu 3. Pomocí něj je zprostředkována akce nahrávání scény (včetně modelů automobilů) do simulátoru V-REP.

Z důvodu čitelnosti je v ukázce kódu vynechána kontrolní část. Před samotným nahráním scény dochází k ověření, zda uživatel vybral okruh a zadal všechny modely automobilů (včetně neuronových sítí) v závislostí na volbě počtu závodníků. K těmto krokům jsou využity funkce `checkCircuit`, `checkCars` a `checkDrivers`. Pokud jsou uvedené podmínky splněny, dochází k části nahrání scény (respektive její vyžádání u nižší vrstvy).

V prvním kroku je vytvořena instance třídy `ConnectorSimulation` jejím konstruktorem. Ten potřebuje dva parametry, kterými jsou počet závodníků (tedy množství modelů, které si přeje uživatel do scény nahrát) a dále pole objektů třídy `File`, pomocí kterých jsou v pozdější fázi jednotlivým vozům nahrávány neuronové sítě. Následně dochází k připojení k serveru (aplikaci V-REP) funkcí `connect` a nahrání scény - `loadScene`. Funkce, která zprostředkuje zmíněné nahrání scény v parametru přijímá její umístění v systému.

Po těchto krocích následuje sekvence krátkodobého spuštění simulace k získání informací o startovních bodech ve scéně a jejímu následnému zastavení. Jakmile jsou tyto informace opatřeny, je zavolána funkce `loadModel` s parametrem objektů třídy `File`, které představují cesty k uživatelem zvoleným modelům. V posledním kroku se spustí vlákno, které je odpovědné za získávání informací ze simulátoru, a to pomocí funkce `start`.

```
btnLoadScene.addActionListener(e -> {
    try {
        cs = new ConnectorSimulation(drivers, netsXml);
        cs.connect();
        cs.loadScene(circuit.getAbsolutePath());
        cs.startSimulation();
        cs.getStartPoints();
        cs.stopSimulation();
        cs.loadModel(cars);
        cs.start();
    } catch (ParserConfigurationException | IOException | SAXException ex) {
        LOGGER.log(Level.SEVERE, ex.toString(), ex);
    }
});
```

Výpis 3: Nahrávání scény a modelů do simulátoru

6.3.2 Ovládání simulace

Další důležitou části třídy `GuiSimulationTab` je ovládání simulace. To je realizováno pomocí tří tlačítek (*start*, *pause* a *stop*), respektive posluchačů na nich zaregistrovaných.

```
btnSimStart.addActionListener(e -> {
    try {
        cs.startSimulation(true);
    } catch (Exception ex) {
        simInstanceNotSet();
    }
});
```

Výpis 4: Spouštění simulace - prezentační vrstva

V rámci spouštění simulace se jedná o zavolání funkce `startSimulation`, což je viditelné ve výpisu 4. Obdobně (zavoláním funkcí jiných) je řešeno jak její pozastavení, tak i zastavení úplné. Ovládání simulace je možné i v programu V-REP, nicméně na uvedená tlačítka externí aplikace je nastavená další funkcionalita (spuštění, respektive zastavení přepočítávání vstupů neuronových sítí), která je nezbytná ke korektnímu běhu zmiňované simulace.

Za předpokladu, že uživatel chce spustit (respektive pozastavit či zastavit) simulaci, jejíž instance ještě nebyla vytvořená, je mu zobrazená varovná hláška.

6.4 Logika

Logická vrstva programu je zastoupená třídami balíčku *connector*. Ten seskupuje všechny nejdůležitější aspekty aplikace. Zmiňované třídy můžeme neformálně zařadit do užších skupin podle jejich využití:

- Třídy běhu a ovládání simulace – v roli hlavního předka je třída `ConnectorBase`. Obsahuje základní logiku k připojování k simulátoru V-REP, práci se scénou či modely a samotnou simulací. Potomky této třídy jsou `ConnectorSimulation` a `ConnectorTest`. Dále do této skupiny řadíme i třídu `ConnectorRace`, pomocí které je v samostatném vláknu spuštěný cyklus k získávání dat ze simulátoru, jejich zpracování a následné odesílání výsledků zpět do programu V-REP. Skupinu uzavírá třída `ConnectorStartPoint`, která je využívaná při lokalizování startovacích bodů ve scéně.

- Třídy automobilu – jedná se o skupinu tříd s prefixem `ConnectorCar`, které jako celek zapouzdřují všechny atributy, funkce a výpočetní moduly potřebné ke správnému přepočtu hodnot získaných na straně simulátoru. Takto přepočítaná data jsou připravená posloužit jako vstupy neuronovým sítím. Výstupy zmiňovaných neuronových sítí jsou rovněž pomocí funkcí těchto tříd konvertovány na hodnoty, s kterými umí pracovat simulátor. Do této skupiny tříd řadíme i jednu bez výše uvedeného prefixu. Jmenovitě se jedná o třídu `ConnectorSensor`.

- Třídy neuronové sítě – jedná se o třídy, pomocí kterých lze načíst naučené neuronové sítě ze souborů XML. Umožňují dotazování nad načtenými sítěmi pomocí implementace metody dopředného šíření.

- Pomocná třída – `ConnectorMath`, která seskupuje funkce pro vedlejší výpočty a kontroly, mezi jinými například výpočet vzdálenosti mezi body v prostoru či výpočet úhlu natočení kol modelu vozidla.

6.4.1 Třídy běhu a ovládání simulace

Bázová třída `ConnectorBase` obsahuje instance tříd komunikační vrstvy, které jsou stěžejní k ovládání simulace. Jedná se o objekty typu `VrepConnection`, `VrepSimulation` a `VrepScene`. Nad těmito objekty jsou vykonávané funkce, které mají za následek změny chování simulátoru V-REP, respektive simulace v něm.

`ConnectorSimulation` a `ConnectorTest` jsou potomky zmiňované bázové třídy. Rozšiřují ji a upřesňují některé její metody. Konstruktor první zmiňované třídy je viditelný ve výpisu 5.

Jak již bylo jednou uvedeno, tento konstruktor přijímá dva parametry, a to počet automobilů které budou do scény importovány a dále pole objektů typu `File` stejné velikosti, reprezentující řídící jednotky (tedy neuronové sítě) takových vozů. V prvním fázi vytváření objektu třídy `ConnectorSimulation` je zavolán konstruktor předka klíčovým slovem `super`. Vynecháním tohoto kroku by nebyla navázána komunikace s aplikací V-REP. Zmiňovaný konstruktor předka totiž spojení se simulátorem zahajuje vytvořením instance třídy nejnižší vrstvy odpovědné za připojení - `VrepConnection`.

```
public ConnectorSimulation (int drivers, File[] netsXml) {
    super();
    race = new ConnectorRace(getConnection(), drivers, netsXml);
    thread = new Thread(race);
    this.drivers = drivers;
    startPoints = new ConnectorStartPoint[drivers];
}
```

Výpis 5: Vytváření simulace - logická vrstva

Třída `ConnectorRace`, jejíž instance je vytvářena ve výše uvedeném konstruktoru, obsahuje metodu `run`, která se spustí při nahrání scény do aplikace V-REP. Ve smyčce je opakována do doby resetu zmiňované scény. V každém svém opakování čte data všech v simulátoru umístěných modelů vozidel pomocí funkce `readData`, zasílá je k vyhodnocení neuronovým sítím funkcí `ask`, a následně posílá zpátky do aplikace V-REP odpověď (respektive nastavuje rychlost vozidla a natočení jeho kol pomocí třídy `ConnectorCar`). Zjednodušená verze popisované funkcionality je zobrazena ve výpisu 6.

```
for (int i~= 0; i~< cars.length; i++) {
    cars[i].readData();
    inputs = setInputs(inputs, i);
    answer = nets[i].ask(inputs);
    cars[i].setWheel((float) answer[0]);
    cars[i].setSpeed((float) answer[1]);
}
```

Výpis 6: Čtení dat a manipulace s odpovědí

6.4.2 Třídy automobilu

Jak už bylo uvedeno, jedná se o skupinu tříd, které zaobalují metody potřebné ke správnému přepočtu hodnot získaných na straně simulátoru. Mimo to takové hodnoty i načítají. Hlavní část této skupiny tvoří třída `ConenctorCar`, která reprezentuje model vozidla. Je složená z objektů jak logické (třídy se stejným prefixem), tak komunikační vrstvy (například motory).

6.4.3 Třídy neuronové sítě

K realizaci implementace tříd neuronových sítí, bylo využito hotového řešení z původního simulátoru *Neural Racer*, respektive jeho klienta. Ten umožňuje načítání neuronových sítí včetně jejich adaptačních množin, trénování či jejich následné testování. K simulaci jízdy vozidel v aplikaci V-REP, přesněji řečeno k ovládání takového vozidla pomocí neuronové sítě, postačuje část implementace, která se věnuje načítání a dotazování. Zbylá část původního kódu tedy není v aplikaci *V-REP Connector* zahrnutá.

6.5 Komunikace

K samotnému spojení externí aplikace se simulátorem V-REP dochází v nejnižší vrstvě aplikace. Třídy této vrstvy obsahují metody *remote* API, které komunikaci zajišťují. Všechny tyto třídy jsou rozpoznatelné podle prefixu `Vrep`. Jsou strukturovány tak, aby byla orientace v nich jednoduchá a intuitivní (například tedy všechny použité funkce z *remote* API potřebné k manipulaci se scénou jsou umístěné ve třídě `VrepScene`).

Při volání metod vzdáleného rozhraní simulátoru jsou často vyžadovány takzvané operační módy (parametry jednotlivých funkcí). Ačkoliv jsou některé specifikovány už v logické vrstvě programu *V-REP Connector*, teprve tady figurují coby vstupy právě do metod *remote* API. V rámci celé aplikace jsou použity tři, a to:

- *simx_opmode_blocking* – blokovací mód. Příkaz s tímto módem v parametru pošle instrukci serveru k vykonání operace a následně čeká na její potvrzení. Použití tohoto módu je doporučeno u funkcí, o jejichž provedení si uživatel potřebuje být jistý a s vrácenou hodnotu (potvrzením) potřebuje dále pracovat.

- *simx_opmode_oneshot* – neblokovací mód. Příkaz v tomto módu pošle instrukci serveru, nečeká ale na odpověď z jeho strany. Z lokálního bufferu se vrátí odpověď poslední instrukce stejného typu. Mód se používá v případech, kdy uživatel potřebuje v simulátoru něco nastavit.

- *simx_opmode_streaming* – rovněž neblokovací mód. Funguje na stejném principu jako mód předešlý, s tím rozdílem, že zaslaný příkaz je na serveru uložen a kontinuálně vykonáván. Většinou se používá v případech, kdy je potřeba získávat stejnou hodnotu po celou dobu běhu simulace.

6.5.1 Navázání spojení

K navázání spojení mezi simulátorem a externí aplikací dochází ve třídě `VrepConnection`. Ta je umístěná v komunikační vrstvě aplikace. Její instance se vytváří ve vrstvě vyšší – logické. Ke korektnímu vytvoření spojení, tedy zmiňované instance popisované třídy, jsou potřeba dva základní parametry, a to lokalizační adresa serveru a specifický port.

V rámci programu *V-REP Connector* je IP adresa nastavená na *localhost*, port na hodnotu 19997. Uvedený port (respektive jeho zmiňovaná hodnota) je takto nastavený záměrně a má své opodstatnění. Vysvětlením je automatické naslouchání spuštěného simulátoru právě na tomto portu. Zdrojový kód používaný pro připojení k simulátoru je viditelný ve výpisu 7.

```
public int connect() {
    vrep.simxFinish(-1);
    clientId = (vrep.simxStart(address, port, true, true, 5000, 5));
    return clientId;
}
```

Výpis 7: Připojení k aplikaci V-REP

V prvním kroku jsou funkcí `simxFinish` s parametrem -1 uzavřena všechna existující spojení, která nebyla předem ukončena uživatelem. Následuje zavolání funkce `simxStart`, kde je nastavená výše uvedená IP adresa a port. Dalšími parametry jsou specifikovány vlastnosti připojení (jedná se o hodnoty doporučené tvůrci aplikace V-REP).

K ovládání simulace lze využít a na straně externí aplikace nastavit jakýkoliv jiný port. Uživatel se pak ale musí smířit s absencí ovládání simulátoru samotného, jelikož spojení je navázáno jen v rámci běžící simulace, kde je v *child* scriptu specifikován stejný port, jak je tomu na straně klienta.

7 Analýza možností rozšíření neuronových sítí

7.1 Rozlišení statické a dynamické překážky

Jedním z úkolů diplomové práce byla možnost specifikace objektů detekovaných automobilem. Přesněji se jedná o rozlišení statických a dynamických překážek. Jelikož simulátor V-REP pracuje s objekty, v jejichž nastavení přímo figurují zmiňované vlastnosti, je takové nastavení využito ke splnění výše popsaného úkolu.

Ke specifikaci typu snímaného objektu dochází na straně simulátoru, a to v *child* scriptech jednotlivých modelů. Ve snímacích segmentech těchto scriptů jsou ovládány senzory přiblížení, které načítají informace o okolí automobilu. Script pak posílá zprávy pomocí řetězce na stranu klienta, tedy do externí aplikace. V nejjednodušší verzi by takový signál obsahoval pouze jednu informaci, a to vzdálenost od snímacího bodu senzoru k detekované překážce. K rozlišení typu snímaného objektu jsou do uvedeného signálu přidány ještě další dvě hodnoty, a to:

- Typ snímaného objektu – je reprezentován hodnotami v rozmezí 0 až 1. Nejnižší hodnota uvedeného intervalu je zasílána v případě, že senzor nedetekuje nic. Hodnotou 0,5 se dává najevo, že senzor snímá statickou překážku. Pokud je signálem zasílána hodnota 1, definuje se ní dynamický objekt v zorném poli senzoru.
- Číselná reprezentace snímaného objektu – tzv. *handle* objektu. Pomocí této numerické hodnoty lze specifikovat objekt v rámci scény. Je zasílána pouze v případě detekce dynamických objektů. Ve zbylých případech permanentně nabývá velikosti -1.

Popisovaný scénář je zobrazen ve výpisu 8. Jedná se o zjednodušenou podobu části scriptu, kterou obsahuje každý model automobilu.

```
if sim.getObjectType(handle)==sim.object_shape_type then
    local r,p=sim.getObjectInt32Parameter(handle,sim.shapeintparam_static)
    if p>0 then
        seg={distance, 0.5, handle}
    else
        seg={distance, 1, -1}
    end
end
```

Výpis 8: Rozlišování dynamických a statických objektů

Pomocí funkce `sim.getObjectType` s parametrem odkazu na snímaný objekt se vyhodnotí, o jaký typ objektu se jedná. V případě, že senzor detekuje objekt typu *shape*, je funkcí `sim.getObjectInt32Parameter` získána informace, zda je takový objekt statický. V závislosti na její vyhodnocení, jsou do proměnné `seg` ukládány informace, popisované výše. Samotná proměnná je později zabalena do řetězce a odeslána jako signál.

7.2 Rozšíření vstupů neuronových sítí

Externí aplikace přijímá řetězce odesílané ze strany simulátoru, upravuje je a posílá k dalšímu zpracování neuronovým sítím. Ty obsahují jako vstupy hodnoty, které reprezentují matici senzorů kolem vozidla a reagují na objekty které se v ní nachází. Bohužel ale nejsou nijak přizpůsobeny tomu, aby odpovídaly jinak na předměty, které se v prostoru pohybují.

Jelikož ze strany simulátoru takové informace přicházejí, nabízí se možnost vstupy neuronových sítí rozšířit. Otázkou zůstává, kolik zmiňovaných vstupů přidat a jaké hodnoty do nich posílat.

Ke korektnímu zpracováni dat přijatých na straně externí aplikace způsobem popisovaným níže musí být určitým způsobem přizpůsobena scéna simulátoru. Předpokládá se tedy, že:

1. Dynamicky simulované objekty ve scéně jsou pouze samotné modely automobilů. Jedná se o jediné pohyblivé modely vůbec.

2. Všechny další překážky, které se ve scéně objevují, jsou statického typu (například stromy či budovy).

Nejjednodušším řešením, kterým by ovšem nebylo docíleno téměř žádného pokroku, by bylo zdvojnásobení počtů vstupů, které reprezentují jednotlivé senzory přiblížení. Na každý zmiňovaný vstup by připadal jeden další, který by specifikoval typ objektu. Neuronová síť by pak mohla reagovat jinak, v závislosti na typu překážky. Stejného výsledku, bez rozšiřování vstupů neuronových sítí, by se dalo docílit i změnou hodnot. U dosavadního řešení se určuje vzdálenost od snímaného objektu hodnotami v rozmezí 0 až 1. Takový interval by se dal rozpůlit. Jeho první polovina by pak mohla reprezentovat vzdálenost statického objektu, druhá zase objektu dynamického. Uvedenými způsoby by se daly neuronové sítě upravit k tomu, aby rozlišovaly pohyblivé a statické překážky.

U zmiňovaných dynamických překážek ale vzniká další problém. Jedná se o rychlost a směr jejich pohybu. Je zřejmé, že vozidlo řízené neuronovou sítí by mělo reagovat jinak na automobil, který se pohybuje po trase stejným směrem (tedy předjetím), a jinak na vůz, který mu jede naproti (úhybným manévrem).

Jelikož je ze strany simulátoru v signálu senzoru zasílána informace o tom, který objekt je detekován, a zároveň se předpokládá, že dynamické modely ve scéně jsou pouze automobily, nic nebrání tomu zjistit na straně klienta informace o voze, který mu je překážkou.

Pomocí informace o rychlosti detekovaného vozu a jeho úhlu natočení od vodící čáry (respektive směru jeho jízdy), se dá neuronové sítí předat informace o chování snímaného modelu automobilu. V tomto případě se už ale samotnému rozšíření vstupů neuronových sítí vyhnout nelze.

8 Závěr

Cílem diplomové práce bylo vytvoření aplikace, která by umožňovala jízdu automobilů řízených neuronovými sítěmi simulovanou pomocí platformy V-REP. Tento úkol byl splněn a jeho výsledkem je program *V-REP Connector*, který umožňuje připojení ke zmiňované platformě a následné ovládání vozů umělou inteligencí.

Implementovaný program představuje podstatné vylepšení simulátoru *Neural Racer*, který je využívaný v předmětu *Neuronové sítě*. Simulace v aplikaci V-REP umožňuje jednodušší a věrnější nastavení fyzických parametrů objektů, tedy hlavně vozidel samotných, ruku v ruce s oku příjemnější vizualizací jízdy po předem vyznačené trase. To sebou ovšem přineslo i určité nevýhody oproti původnímu simulátoru, jako je například nemožnost programově měnit trať anebo nutnost předzpracování některých hodnot na straně simulátoru v programovacím jazyce Lua.

Jednou z časově nejnáročnějších pasáží bylo důkladné nastudování simulátoru V-REP. Bez znalosti funkcionality a s absencí pochopení základních principů jeho běhu by byl hlavní cíl závěrečné práce nesplnitelný. Trať a automobil totiž tvoří dva nejdůležitější prvky, pomocí kterých jsou získávané data, na které posléze reagují neuronové sítě. Je tedy zřejmé, že chyba při načítání informací by vedla k nekorektnímu vyhodnocení výsledků, které jsou k následnému průběhu simulace stěžejní.

Do budoucna by bylo možné rozšířit neuronové sítě používané k řízení jízdy. Aplikace totiž v aktuálním stádiu umí poskytovat data, se kterými doposud zmiňované neuronové sítě nepočítaly, jako je například rozlišování typů překážek před vozem. Jako další možné vylepšení se jeví věrnější vymodelování vozidla, včetně přesného nastavení jeho fyzikálních vlastností a parametrů.

Literatura

[1] VONDRÁK, Ivo. *Neuronové sítě* [online]. Vysoká škola báňská - Technická univerzita Ostrava, 1994 [cit. 2019-04-04]. Dostupné z: `http://vondrak.cs.vsb.cz/download/Neuronove_site.pdf`

[2] COPPELIA ROBOTICS. *V-REP User Manual* [online]. 2019 [cit. 2019-04-08]. Dostupné z: `http://www.coppeliarobotics.com/helpFiles`

[3] Bullet Real-Time Physics Simulation. *Home of Bullet and PyBullet: physics simulation for games, visual effects, robotics and reinforcement learning.* [online]. [cit. 2019-04-11]. Dostupné z: `https://pybullet.org`

[4] Open Dynamics Engine. *Open Dynamics Engine* [online]. © 2001-2004 Russell L. Smith [cit. 2019-04-11]. Dostupné z: `https://www.ode.org`

[5] Newton Dynamics. *Newton Dynamics* [online]. © 2003-2011 [cit. 2019-04-11]. Dostupné z: `http://newtondynamics.com`

[6] PUC-RIO. *Lua 5.3 Reference Manual* [online]. © 2015–2018 Lua.org, PUC-Rio [cit. 2019-04-12]. Dostupné z: `https://www.lua.org/manual/5.3`

[7] WindowBuilder. *WindowBuilder* [online]. © Eclipse Foundation [cit. 2019-04-17]. Dostupné z: `https://www.eclipse.org/windowbuilder`

[8] ORACLE. *Package javax.swing (Java Platform SE 8)* [online]. © 1993-2019 Oracle [cit. 2019-04-17]. Dostupné z: `https://docs.oracle.com/javase/8/docs/api`

Přílohy

I. Příloha na CD

Elektronická příloha na disku CD obsahuje složky *vrc_jar* a *vrc_src*. V první uvedené se nachází spustitelný soubor programu *V-REP Connector*, včetně předem vytvořených tras a vymodelovaných automobilů. Druhá obsahuje zdrojové kódy zmiňované aplikace.